UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INFORMÁTICA

Diseño y simulación de un robot asistente para personas invidentes

Proyecto Fin de Carrera

Ingeniería Técnica Telecomunicaciones. Sistemas de Telecomunicaciones

Autora: Paula Berrio Martínez
Tutor: Raúl Arrabales Moreno

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1.INTRODUCCIÓN

1. INTRODUCCIÓN

Según algunos autores la robótica está a punto de vivir una revolución similar a la que experimentó la computación o la telefonía en décadas pasadas y el papel de España en este ámbito es el de liderar algunos campos científicos y tecnológicos estratégicos [*1*]. Además, se espera que el acercamiento de la robótica a la sociedad mejore la calidad de vida de los ciudadanos.

La aplicación Industrial de la Robótica no necesita introducción ya que sus resultados son bien conocidos. La Robótica Industrial ha dado lugar, entre otras cosas, a procesos de producción mucho más eficientes, a una mayor calidad de los productos, etc. Todos estos elementos aumentan la competitividad de una industria (país) frente a sus similares. Desde finales de los años 80 y principios de los 90 un nuevo enfoque en la Robótica se ha consolidado. Este nuevo tipo de Robótica se denomina Robótica Autónoma.

Los Robots Autónomos (RA) son sistemas completos que operan eficientemente en entornos complejos sin necesidad de estar constantemente guiados y controlados por operadores humanos. Una propiedad fundamental de los RA es la de poder reconfigurarse dinámicamente para resolver distintas tareas según las características del entorno impuestas en un momento dado. Se hace énfasis en que son sistemas completos que perciben y actúan en entornos dinámicos y parcialmente impredecibles, coordinando interoperaciones entre capacidades complementarias de sus componentes. La funcionalidad de los RA es muy amplia y variada desde algunos RA que trabajan en entornos inhabitables, a otros que están diseñados para asistir a personas con discapacidad.

Una característica fundamental y diferenciadora de la robótica avanzada será la estrecha colaboración de los robots con los humanos, tanto en el campo industrial como en el doméstico. Esta nueva sociedad robotizada implicara un importante cambio en el modo y calidad de vida de los ciudadanos.

La robótica tiene como intención final complementar o sustituir las funciones de los humanos, alcanzando, en algunos sectores, aplicaciones masivas. Por otro lado, hay que destacar que la robótica puede ofrecer unos grandes beneficios

sociales, resolviendo problemas cotidianos en todos los sectores y edades de la población, mejorando la calidad de vida de los ciudadanos mediante la reducción de las horas de trabajo y de los riesgos laborales. También puede aportar beneficios económicos aumentando la competitividad de las empresas, dinamizando la creación de nuevas empresas y nuevos modelos de negocio y profesiones.

Este proyecto surge cuando una vez conocido el campo y ámbito de la robótica y sus aplicaciones actuales, se intenta dar un uso práctico para solventar un problema que está presente en nuestra sociedad.

Todos los días, en nuestras ciudades, se ven personas invidentes que con mucho esfuerzo se afrontan al mundo con una gran limitación. Estas personas cada vez más están más integradas en nuestra sociedad gracias a las facilidades que poco a poco se van instalando a nuestro alrededor, y el objetivo en todo momento es que puedan llevar una vida lo más normal posible.

El envejecimiento medio de la población hace que los robots asistentes jueguen un papel importante en los próximos años. La población en un futuro seguramente tendrá más salud que la de ahora pero precisará de mayor asistencia debido a la mayor esperanza de vida. Por otra parte se ha producido en los últimos años cambios en la forma de vida, en los que muchas personas viven solas, o los dos cónyuges de las familias trabajan por lo que cada vez se tiene menos tiempo para realizar las tareas del hogar. Estos hechos hace que se precisen casas cada vez más robots que sean capaces de realizar tareas del hogar: limpieza, traernos o llevarnos cosas, cocinar, lavar, planchar, etc.

La integración de Robots asistentes requiere de desarrollo de sistemas avanzados de interacción y cooperación con los humanos que precisan de sistemas mecánicos más ligeros que no supongan un peligro para las personas. Estos sistemas requieren la integración de nuevos materiales, control, electrónica y diseño. Además se deben mejorar los interfaces de usuario que permitan a las personas sin conocimientos de robótica controlar e interactuar con los Robots de forma Natural. Para lograr estos objetivos es preciso dotar a los Robots de capacidades cognitivas que conlleven un mayor esfuerzo para mejorar el aprendizaje y razonamiento

El objetivo del presente proyecto es proporcionar un servicio guía a una persona invidente dentro de un hotel, todo ello en un entorno simulado. Un robot Pioneer 3 DX se encargará de guiar a esta persona por todo el complejo. El robot

será capaz de asimilar las órdenes de la persona para desplazarse hasta un determinado punto.

Dado que no se disponen de medios ilimitados, habrá que suponer y acotar el problema de una forma más sencilla que se explicará en los siguientes puntos.

1.1. Objetivos. Resolución de un problema de Ingeniería

El objetivo principal que se pretende alcanzar es la resolución del ya conocido problema de generar un camino dentro de un entorno conocido, en este caso un hotel. El robot deberá moverse dentro del mismo y guiar a una persona invidente por las instalaciones. Para ello será necesario dominar el lenguaje de programación que se vaya a utilizar, el entorno de desarrollo, que en este caso será Microsoft Robotics Developer Studio, herramientas de diseño en 3D para generar el entorno y aplicar un algoritmo de caminos mínimos para recorrer el hotel por una ruta establecida. Estos objetivos se explican con más detalle en los puntos siguientes.

- **Análisis de la plataforma de desarrollo MRDS.**

Será necesario el manejo de Microsoft Robotics Developer Studio para llevar a cabo el proyecto, ya que es el pilar fundamental del desarrollo de la aplicación.

- **Análisis de posibles herramientas de desarrollo.**

Se analizarán dos herramientas para el desarrollo del código de la aplicación, VPL (Visual Programming Language), entorno que ofrece un modelo de programación gráfico y que permite una programación convencional, y C#, lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. Una vez analizadas estas dos herramientas se utilizará para el desarrollo la más adecuada para este caso en concreto.

- **Creación de un entorno virtual para simulación.**

Para llevar a cabo la simulación, es necesaria la recreación de un hotel, para posteriormente hacer que el robot se mueva por él. Será un entorno sencillo de una planta baja con varias habitaciones con sus baños, cafetería y hall, creado con el entorno de simulación visual de Microsoft.

- **Manejo de SolidWorks para la creación de entidades 3D simuladas.**

Para realizar el mobiliario del hotel, será necesario su diseño previo en 3D, ya que el entorno de simulación visual que proporciona MRDS únicamente tiene entidades sencillas. Algunas como las paredes se podrán adecuar usando las existentes, pero la mayoría del mobiliario será creado con SolidWorks.

- **Creación de una aplicación orientada a servicios.**

MRDS es una plataforma orientada a servicios que incluye servicios en tiempo de ejecución basados en .NET. Para este proyecto se tendrán que integrar varios servicios para que el conjunto sea una solución óptima del objetivo propuesto. Para ello, los diferentes servicios tendrán que ser analizados para su comprensión y correcta integración en el proyecto.

- **Diseño de un sistema de control para el robot.**

El esquema de movimiento del robot que se realizará en este trabajo consistirá en secuencias sencillas compuestas por movimientos rectos y giros por una ruta prefijada, invisible en la simulación. El robot únicamente se moverá por los diferentes caminos para ir de unas habitaciones a otras. Para implementar este movimiento, será necesario utilizar un algoritmo de caminos mínimos para no hacer ciclos en la ruta y desplazarse por el entorno por el camino más corto**.**

2. ESTADO DEL ARTE

2. ESTADO DEL ARTE

Los sectores a los que actualmente está orientada la robótica son muy amplios empezando por la industria manufacturera (automóvil y máquina herramienta) hasta la exploración de ambientes hostiles tales como entornos submarinos y espacio. No obstante, la "robotización" no sólo tiene fines industriales, sino también múltiples aplicaciones sociales, tales como asistencia personal, medicina, limpieza, inspección y mantenimiento de infraestructuras. De hecho, la robótica actual se divide en dos grandes áreas, la robótica industrial y la robótica de servicio, entendiéndose esta última en un sentido amplio: servicios personales y a la sociedad. Aunque la robótica industrial está bien establecida desde hace varias décadas y la de servicios en una fase incipiente, ambas presentan grandes posibilidades de investigación y desarrollo que dan lugar a la robótica avanzada [*2*].

La robótica de servicio es un campo emergente, pero con un gran potencial de crecimiento. Sus aplicaciones se dividen en servicios personales (asistencia a personas mayores, discapacitados y niños, acompañante y/o sirviente personal, limpieza y seguridad doméstica, etc.) y servicios profesionales (limpieza de calles, vigilancia urbana, mantenimiento e inspección de infraestructuras, compañero de trabajo, medicina, construcción, agricultura, etc.). La mayoría de los sectores y aplicaciones citadas cuentan con un bajo o muy bajo nivel de automatización, ocupando un gran número de trabajadores en actividades tediosas y en algunos casos peligrosas. Además el continuo envejecimiento de la población, sin medidas efectivas para su cuidado y ocio, hace cada vez más necesarios el desarrollo de robots para este sector de la población.

2.1. Contexto histórico

El inicio de la robótica actual puede fijarse en la industria textil del siglo XVIII, cuando Joseph Jacquard inventa en 1801 una máquina textil programable mediante tarjetas perforadas. La revolución industrial impulsó el desarrollo de estos agentes mecánicos, entre los cuales se destacaron el torno mecánico motorizado de Babbitt (1892) y el mecanismo programable para pintar con espray de Pollard y Roselund (1939). Además de esto durante los siglos XVII y XVIII en Europa fueron construidos muñecos mecánicos muy ingeniosos que tenían algunas características de robots. Jacques de Vauncansos construyó varios músicos de tamaño humano a mediados del siglo XVIII. Esencialmente se trataba de robots mecánicos diseñados para un propósito específico: la diversión.

En 1805, Henri Maillardert construyó una muñeca mecánica que era capaz de hacer dibujos. Una serie de levas se utilizaban como “el programa” para el dispositivo en el proceso de escribir y dibujar. Estas creaciones mecánicas de forma humana deben considerarse como inversiones aisladas que reflejan el genio de hombres que se anticiparon a su época.

La palabra robot se empleó por primera vez en 1920 en una obra de teatro llamada "R.U.R." o "Los Robots Universales de Rossum" escrita por el dramaturgo checo Karel Capek. La trama era sencilla: el hombre fabrica un robot luego el robot mata al hombre. Muchas películas han seguido mostrando a los robots como máquinas dañinas y amenazadoras. La palabra checa “Robota” significa servidumbre o trabajador forzado, y cuando se tradujo al inglés se convirtió en el término robot.

Entre los escritores de ciencia ficción, Isaac Asimov contribuyó con varias narraciones relativas a robots, comenzó en 1939, a él se atribuye el acuñamiento del término Robótica. La imagen de robot que aparece en su obra es el de una máquina bien diseñada y con una seguridad garantizada que actúa de acuerdo con tres principios.

Estos principios fueron denominados por Asimov las Tres Leyes de la Robótica y son:

1.- Un robot no puede actuar contra un ser humano o, mediante la inacción, que un ser humano sufra daños.

2.- Un robot debe de obedecer las órdenes dadas por los seres humanos,

salvo que estén en conflictos con la primera ley.
3.- Un robot debe proteger su propia existencia, a no ser que esté en conflicto con las dos primeras leyes.

Consecuentemente todos los robots de Asimov son fieles sirvientes del ser humano, de ésta forma su actitud contraviene a la de Kapek. Inicialmente, se definía un robot como un manipulador reprogramable y multifuncional diseñado para trasladar materiales, piezas, herramientas o aparatos a través de una serie de movimientos programados para llevar a cabo una variedad de tareas. El desarrollo en la tecnología, donde se incluyen las poderosas computadoras electrónicas, los actuadores de control retroalimentados, transmisión de potencia a través de engranes, y la tecnología en sensores han contribuido a flexibilizar los mecanismos autómatas para desempeñar tareas dentro de la industria. Son varios los factores que intervienen para que se desarrollaran los primeros robots en la década de los 50's.

La investigación en inteligencia artificial desarrolló maneras de emular el procesamiento de información humana con computadoras electrónicas e inventó una variedad de mecanismos para probar sus teorías. Las primeras patentes aparecieron en 1946 con los muy primitivos robots para traslado de maquinaria de Devol. También en ese año aparecen las primeras computadoras: J. Presper Eckert y John Maulchy construyeron el ENAC en la Universidad de Pensilvania y la primera máquina digital de propósito general se desarrolla en el MIT (Massachusetts Institute of Technology).

En 1954, Devol diseña el primer robot programable y acuña el término "autómata universal", que posteriormente recorta a Unimation. Así llamaría Engleberger a la primera compañía de robótica. La comercialización de robots comenzaría en 1959, con el primer modelo de la Planet. En 1964 se abren laboratorios de investigación en inteligencia artificial en el MIT, el SRI (Stanford Research Institute) y en la universidad de Edimburgo. Poco después los japoneses que anteriormente importaban su tecnología robótica, se sitúan como pioneros del mercado. Otros desarrollos Importantes en la historia de la robótica fueron:

En 1960 se introdujo el primer robot "Unimate", basada en la transferencia de articulaciones programada de Devol. Utilizan los principios de control numérico para el control de manipulador y era un robot de transmisión hidráulica.

En 1961 Un robot Unimate se instaló en la Ford Motors Company para atender una máquina de fundición de troquel.

En 1966 Trallfa, una firma noruega, construyó e instaló un robot de pintura por pulverización.

En 1971 El "Standford Arm", un pequeño brazo de robot de accionamiento eléctrico, se desarrolló en la Standford University.

En 1973 Se desarrolló en SRI el primer lenguaje de programación de robots del tipo de computadora para la investigación con la denominación WAVE. Fue seguido por el lenguaje AL en 1974. Los dos lenguajes se desarrollaron posteriormente en el lenguaje VAL comercial para Unimation por Víctor Scheinman y Bruce Simano.

En 1978 Se introdujo el robot PUMA (Programmable Universal Machine for Assambly) para tareas de montaje por Unimation, basándose en diseños obtenidos en un estudio de la General Motors.

En 1980 Un sistema robótico de captación de recipientes fue objeto de demostración en la Universidad de Rhode Island. Con el empleo de visión de máquina el sistema era capaz de captar piezas en orientaciones aleatorias y posiciones fuera de un recipiente.

Actualmente, el concepto de robótica ha evolucionado hacia los sistemas móviles autónomos, que son aquellos que son capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión. El primer robot móvil de la historia, pese a sus muy limitadas capacidades, fue ELSIE (Electro-Light-Sensitive Internal-External), construido en Inglaterra en 1953. ELSIE se limitaba a seguir una fuente de luz utilizando un sistema mecánico realimentado sin incorporar inteligencia adicional.

En 1968, apareció SHACKEY del SRI (standford Research Institute), que estaba provisto de una diversidad de sensores así como una cámara de visión y sensores táctiles y podía desplazarse por el suelo. El proceso se llevaba en dos computadores conectados por radio, uno a bordo encargado de controlar los motores y otro remoto para el procesamiento de imágenes.

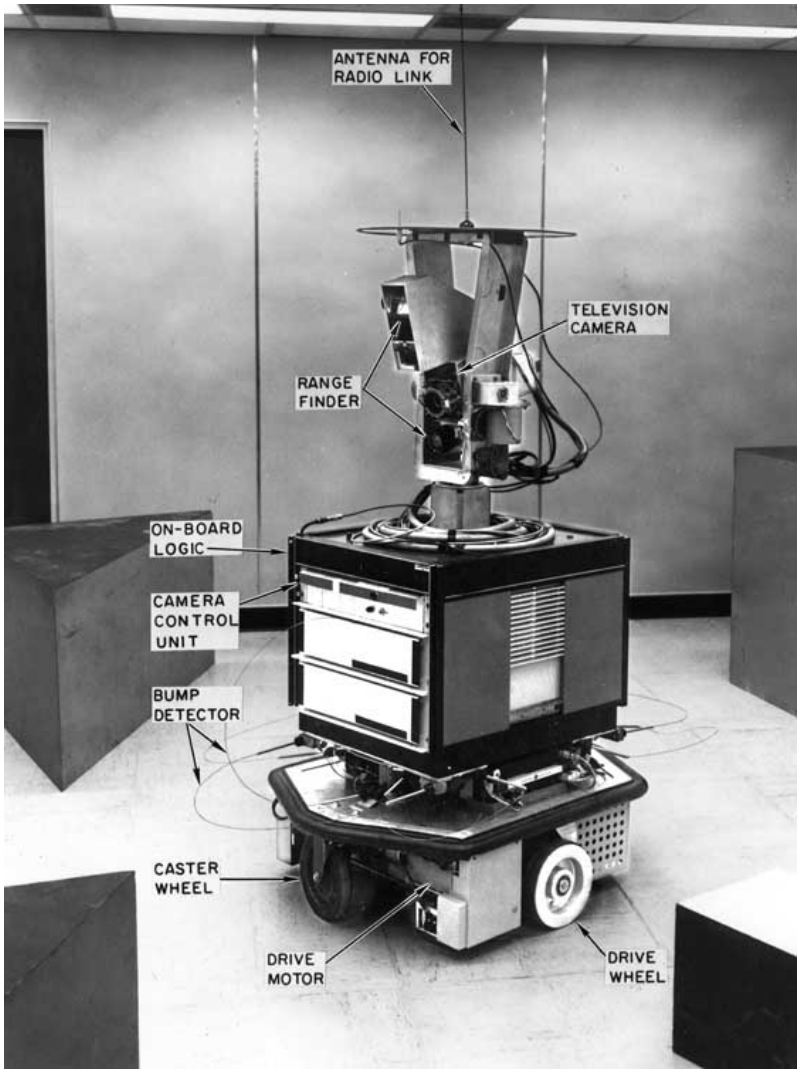

Figura 1. Robot Shackey

En los setenta, la NASA inicio un programa de cooperación con el Jet Propulsion Laboratory para desarrollar plataformas capaces de explorar terrenos hostiles. El primer fruto de esta alianza seria el MARS-ROVER, que estaba equipado con un brazo mecánico tipo STANFORD, un dispositivo telemétrico láser, cámaras estéreo y sensores de proximidad.

Figura 2. Robot Mars-Rover

En los ochenta aparece el CART del SRI que trabaja con procesado de imagen estéreo, más una cámara adicional acoplada en su parte superior.

También en la década de los ochenta, el CMU-ROVER de la Universidad Carnegie Mellon incorporaba por primera vez una rueda timón, lo que permite cualquier posición y orientación del plano.

En la actualidad, la robótica se debate entre modelos sumamente ambiciosos, como es el caso del IT, diseñado para expresar emociones, el COG, también conocido como el robot de cuatro sentidos. También es muy conocido el famoso SOUJOURNER o el LUNAR ROVER, vehículo con control remoto, y otros mucho más específicos como el CYPHER, un helicóptero robot de uso militar, el guardia de tráfico japonés ANZEN TARO o los robots mascotas de Sony.

Figura 3. Robot mascota de Sony

En el campo de los robots antropomorfos (androides) se debe mencionar el P3 de Honda que mide 1.60m, pesa 130 Kg. y es capaz de subir y bajar escaleras, abrir puertas, pulsar interruptores y empujar vehículos.

En general la historia de la robótica la podemos clasificar en cinco generaciones (división hecha por Michael Cancel, director del Centro de Aplicaciones Robóticas de Science Application Inc. En 1984). Las dos primeras, ya alcanzadas en los ochenta, incluían la gestión de tareas repetitivas con autonomía muy limitada. La tercera generación incluiría visión artificial, en lo cual se ha avanzado mucho en los años ochenta y noventa. La cuarta incluye movilidad avanzada en exteriores e interiores y la quinta entraría en el dominio de la inteligencia artificial más general en lo cual se está trabajando actualmente.

2.1.1. La robótica en la actualidad

Los robots son usados hoy en día para llevar a cabo tareas sucias, peligrosas, difíciles, repetitivas o embotadas para los humanos. Esto usualmente toma la forma

de un robot industrial usado en las líneas de producción. Otras aplicaciones incluyen la limpieza de residuos tóxicos, exploración espacial, minería, búsqueda y rescate de personas y localización de minas terrestres.

La manufactura continúa siendo el principal mercado donde los robots son utilizados. En particular, robots articulados (similares en capacidad de movimiento a un brazo humano) son los más usados comúnmente. Las aplicaciones incluyen soldado, pintado y carga de maquinaria. La Industria automotriz ha tomado gran ventaja de esta nueva tecnología donde los robots han sido programados para reemplazar el trabajo de los humanos en muchas tareas repetitivas. Existe una gran esperanza, especialmente en Japón, de que el cuidado del hogar para la población de edad avanzada pueda ser llevado a cabo por robots.

Recientemente, se ha logrado un gran avance en los robots dedicados a la medicina, con dos compañías en particular, *Computer Motion* e *Intuitive Surgical*, que han recibido la aprobación regulatoria en América del Norte, Europa y Asia para que sus robots sean utilizados en procedimientos de cirugía invasiva mínima. Desde la compra de Computer Motion (creador del robot Zeus) por Intuitive Surgical, se han desarrollado ya dos modelos de robot daVinci por esta última. En la actualidad, existen más de 800 robots quirúrgicos daVinci en el mundo, con aplicaciones en Urología, Ginecología, Cirugía general, Cirugía Pediátrica, Cirugía Torácica, Cirugía Cardíaca y ORL. La automatización de laboratorios también es un área en crecimiento. Aquí, los robots son utilizados para transportar muestras biológicas o químicas entre instrumentos tales como incubadoras, manejadores de líquidos y lectores. Otros lugares donde los robots están reemplazando a los humanos son la exploración del fondo oceánico y exploración espacial. Para esas tareas, se utilizan generalmente robots de tipo artrópodo. Mark W. Tilden del Laboratorio Nacional de los Álamos se especializa en robots económicos de piernas dobladas pero no empalmadas [*4*] mientras que otros buscan crear la réplica de las piernas totalmente rígidas de los cangrejos.

Robots alados experimentales y otros ejemplos que explotan el biomimetismo también están en fases previas. Se espera que los así llamados nanomotores y cables inteligentes simplifiquen drásticamente el poder de locomoción, mientras que la estabilización en vuelo parece haber sido mejorada substancialmente por giroscopios extremadamente pequeños. Un impulsor muy significante de este tipo de trabajo es el desarrollar equipos de espionaje militar.

También, la popularidad de series de televisión como *Robot Wars* y *Battlebots*, de batallas estilo sumo entre robots, el éxito de las Bomba Inteligente y UCAVs en los conflictos armados, los comedores de pasto "gastrobots" en Florida, y la creación de un robot comedor de lingotes en Inglaterra, sugieren que el miedo a las formas de vida artificial haciendo daño, o la competencia con la vida salvaje, no es una ilusión.

Dean Kamen, fundador de FIRST y de la Sociedad Americana de Ingenieros Mecánicos (ASME), ha creado una competición Robótica multinacional que reúne a profesionales y jóvenes para resolver un problema de diseño de ingeniería de una manera competitiva. En 2003 contó con más de 20.000 estudiantes en más de 800 equipos en 24 competiciones. Los equipos vienen de Canadá, Brasil, Reino Unido y Estados Unidos. A diferencia de las competiciones de los robots de lucha sumo, que tienen lugar regularmente en algunos lugares, o las competencias de "Battlebots" transmitidas por televisión, estas competiciones incluyen la creación de un robot.

Los robots parecen estar abaratándose y empequeñeciéndose, todo relacionado con la miniaturización de los componentes electrónicos que se utilizan para controlarlos. También, muchos robots son diseñados en simuladores mucho antes de que sean construidos e interactúen con ambientes físicos reales.

2.1.2. La Robótica en España

Se puede afirmar que España tiene un importante potencial en investigación en robótica (más de 60 grupos), siendo en algunos de ellos y en algunas líneas de investigación pioneros y líderes, tanto en el ámbito europeo como mundial. El peso de la robótica española es importante e incluso superior a países con mayor poder económico que el nuestro, siendo el tercer país por número de grupos en la Red Europea de Robótica [*5*].

España ocupa un lugar relevante en la robótica industrial, estando en el 7º lugar en el mundo y el 4º en Europa por número de robots instalados, con cerca de 22.000 unidades, bastante por encima del Reino Unido y muy cerca de Francia. Del

mismo modo, si se toma como indicador la tasa de robots instalados por cada 10.000 trabajadores en la industria manufacturera, España de encuentra en un destacado lugar, por detrás solamente de Japón, Corea, Alemania, Italia, Suecia y Finlandia [*6*].

2.2. Introducción a los Robots Autónomos

Estos robots presentan unas características particulares que permiten que se puedan aprovechar en gran medida de técnicas de aprendizaje para mejorar su desempeño.

Los robots autónomos son entidades físicas (o simuladas) con capacidad de percepción sobre un entorno y que actúan sobre el mismo en base a dichas percepciones, sin supervisión directa de otros agentes.

En muchas ocasiones se asimila el término "robot autónomo" con el de "robot móvil", pero en realidad son términos diferentes. Un robot autónomo suele ser móvil, entendiendo por móvil que no se encuentra fijado a una posición y puede desplazarse por su entorno, pero no hay nada que en principio obligue a ello. A la inversa, un robot móvil no es necesariamente autónomo: existen multitud de robots móviles que son tele-operados en mayor o menor medida, con lo que contravienen la definición.

Las características principales de los robots autónomos son las siguientes:

Están **situados**: Cuando se dice que un robot autónomo percibe un entorno y actúa sobre él, no se hace a la ligera. El robot está literalmente inmerso en el entorno, esto es, el robot no actúa sobre abstracciones o modelos, sino directamente sobre la realidad material.

Son **entidades corpóreas**: Los robots operan sobre el mundo físico; su experiencia del mundo y sus acciones sobre el mismo se producen de forma directa haciendo uso de sus propias capacidades físicas.

2.2.1. Paradigmas en la Robótica

En las ciencias de la computación y en la ingeniería frecuentemente se hace uso del término "paradigma". Resulta interesante examinar la etimología de esta palabra, así como el significado con el que se usa en la actualidad. Paradigma proviene del griego *paradeigma*, y su significado era originalmente "ejemplo ilustrativo". En particular se usaba para denotar un enunciado modelo que mostraba todas las inflexiones de una palabra.

En la robótica esta definición se adapta mejor que en cualquier otra disciplina, ya que en este campo un paradigma es literalmente una manera de conseguir que los robots tengan una forma de ver el mundo.

Los paradigmas principales en las arquitecturas de control de robots son el jerárquico, el reactivo y el híbrido. En la actualidad el paradigma más empleado es el híbrido, ya que con él se consigue lo mejor de los mundos deliberativo y reactivo sin añadir una mayor complejidad al modelo. Sin embargo para construir un robot no se puede quedar al nivel de los paradigmas, puesto que son demasiado abstractos para una implementación.

Para implementar con éxito un robot se debe diseñar una arquitectura que esté basada en los principios del paradigma escogido, y programar los comportamientos deseados del robot sobre esa arquitectura. Cada una de estas tareas es lo suficientemente compleja como para dedicarle una amplia atención. Aquí se hablará la segunda de ellas, esto es, cuando se tiene una arquitectura implantada y se debe añadir los comportamientos del robot, ¿cómo se puede hacer esto?

Existen diversas metodologías y lenguajes para llevar a cabo esta implementación de comportamientos. Uno de los ejemplos más famosos históricamente es el desarrollado por Nilsson y su equipo, que diseñaron el robot Shakey, el primer robot autónomo construido. Los comportamientos de este robot estaban codificados utilizando el lenguaje Strips (Stanford Research Institute Problem Solver), creado también por el equipo de Nilsson para este proyecto. Este lenguaje se adapta muy bien a la codificación de comportamientos para robots deliberativos, aunque posteriormente también ha venido siendo aplicado en otros entornos.

Existen otras aproximaciones, como los sistemas orientados por objetivos, también llamados teleo-reactivos [*3*], que se basan en conjuntos de reglas de producción que se aplican en un orden concreto. El nombre de "orientados por objetivos" se debe a que con esta arquitectura unas reglas llevan al sistema a un estado que le permite aplicar reglas de más alto nivel, que son las que conducen al robot a su objetivo final.

Este ejemplo está mucho más cercano al paradigma reactivo, ya que las reglas pueden ordenarse en base su capacidad de garantizar la supervivencia del robot, es decir, las primeras de la lista serían las que permiten que el robot no colisione con objetos cercanos, las siguientes serían las que le permiten navegar por el entorno, etc.

2.2.2. La importancia del aprendizaje en la Robótica

En cualquier caso, los paradigmas y arquitecturas que se han comentado, por sí solos adolecen de la capacidad de dotar al robot de herramientas para sobrevivir en un entorno abierto y de condiciones cambiantes. Es por ello por lo que se considera de tanta importancia el poder proporcionar a los robots mecanismos de aprendizaje.

2.3. Técnicas de Navegación Global.

A continuación se introducen los conceptos básicos y técnicas utilizadas para el desarrollo de técnicas de navegación local en robots autónomos móviles. En primer lugar se estudian las estructuras de datos que se utilizan típicamente en dichos desarrollos y después se repasan los mecanismos más comunes de generación de trayectorias. El trabajo realizado en este proyecto se basa en estos conceptos.

2.3.1. Conceptos Teóricos y Estructuras de Datos

La teoría de grafos es a menudo el marco conceptual usado para la representación del problema de navegación global en robots móviles. A continuación se describen brevemente los principales aspectos de esta teoría haciendo hincapié en las estructuras de datos que habrán de manejarse en el desarrollo realizado en el presente proyecto.

- **Grafos**

Los grafos están compuestos por una serie de nodos y aristas como si fueran árboles, pero con la salvedad de que no hay reglas para las conexiones entre los nodos. Con los grafos, no existe el concepto de nodo raíz, ni el concepto de padres e hijos. Más bien, un grafo es una colección de nodos.

En la figura 4, se muestra un ejemplo de grafos. Notar que los grafos tienen una serie de nodos que están desconectados de otros nodos. Por ejemplo, el grafo (a) tiene dos ramas distintas de nodos no conectados. Los grafos pueden contener ciclos, como es el caso (b). Un ciclo es la ruta de v1 a v2 a v4 y vuelta a v1. Otro sería de v1 a v2 a v3 a v5 a v4 y vuelta a v1. El grafo (c) por ejemplo, no tiene ningún ciclo, por tanto es un árbol.

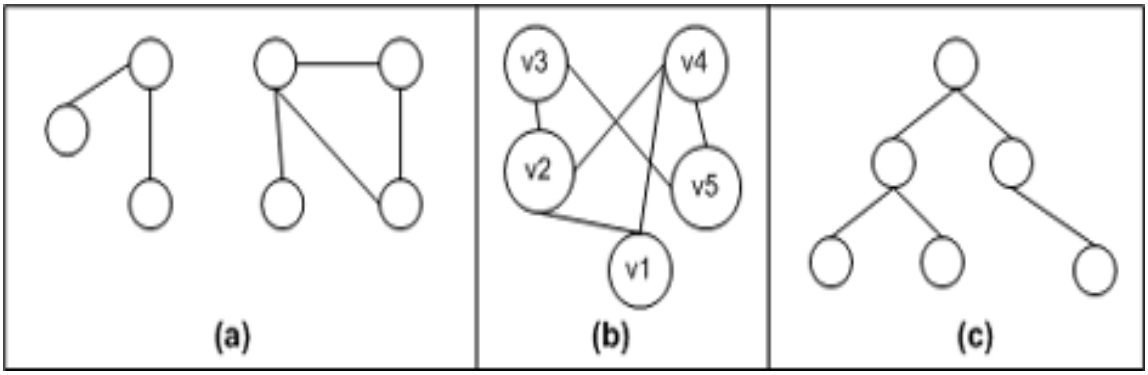

Figura 4. Ejemplo de Grafos

Algunos problemas del mundo real se pueden modelar usando grafos. Por ejemplo, motores de búsqueda como Google modelan Internet como un grafo, donde las páginas Web son nodos del grafo y los enlaces entre páginas son aristas. Programas como Microsoft MapPoint que puede generar rutas de carreteras entre una ciudad y otra usa grafos, modelando las ciudades como nodos del grafo y las carreteras entre ciudades como aristas.

- **Diferentes clases de aristas**

Un grafo, en términos simples es una colección de nodos y aristas, pero hay una diferencia de tipos de aristas:

- Dirigidas vs. No dirigidas.

  Con peso y sin peso.

Cuando hablamos sobre usar grafos para modelar un problema, es importante indicar la clase de grafos con la que se va a trabajar. ¿Es un grafo cuyas aristas son dirigidas y con peso o son no dirigidas y sin peso? En el siguiente punto se explicarán las diferencias entre aristas dirigidas y no dirigidas, con peso y sin peso.

- **Aristas dirigidas vs. Aristas no dirigidas**

Las aristas de un grafo proporcionan las conexiones entre un nodo y otro. Por defecto, una arista se asume que es bidireccional. Esto es, si existe una arista entre los nodos v y u, se asume que se puede ir de v a u y de y a v. Los grafos con aristas bidireccionales son llamados grafos no dirigidos porque no hay ninguna dirección implícita en sus aristas.

Para algunos problemas, una arista podría deducir una conexión de dirección única de un nodo a otro. Por ejemplo, cuando se modela un grafo como Internet, un hipervínculo de una página Web v a una página u podría implicar que la arista entre

v y u puede ser unidireccional, es decir, se podría navegar de v a u pero no de u a v. Los grafos que usan aristas unidireccionales se llaman grafos dirigidos.

Cuando se dibuja un grafo, las aristas bidireccionales son dibujadas como una línea recta, como se muestra en la figura 4.

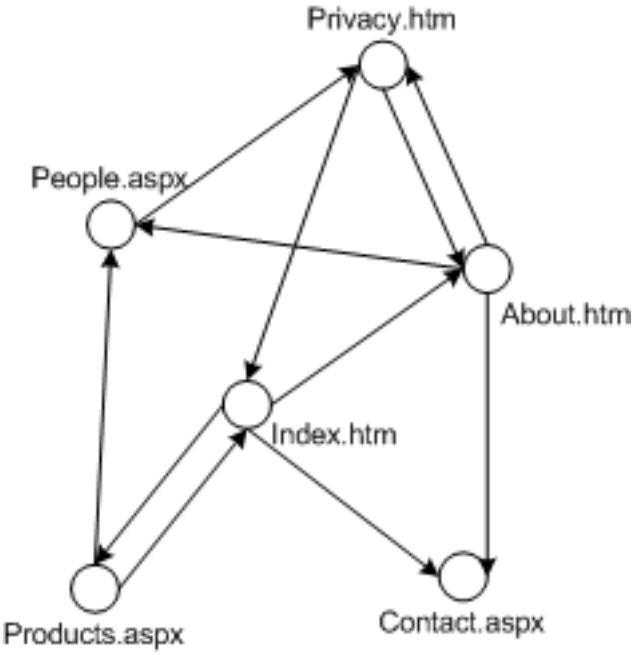

Figura 5. Modelo de páginas de un sitio web.

- **Aristas con peso vs. Aristas sin peso**

Típicamente los grafos se usan para modelar una colección de “cosas” y la relación entre esas “cosas”. Por ejemplo, el grafo de la figura 5 modela las páginas en un sitio web y sus hipervínculos. A veces, es importante asociar un coste a la conexión entre un nodo y otro.

Un mapa puede ser modelado fácilmente como un grafo, con las ciudades como nodos y las carreteras que conectan las ciudades como aristas. Si se quiere determinar la menor distancia entre una ruta de una ciudad a otra, primeramente es

necesario asignar un coste de ir de una ciudad a otra. La solución más lógica sería de asignar a cada arista un peso, como los kilómetros que hay entre las ciudades.

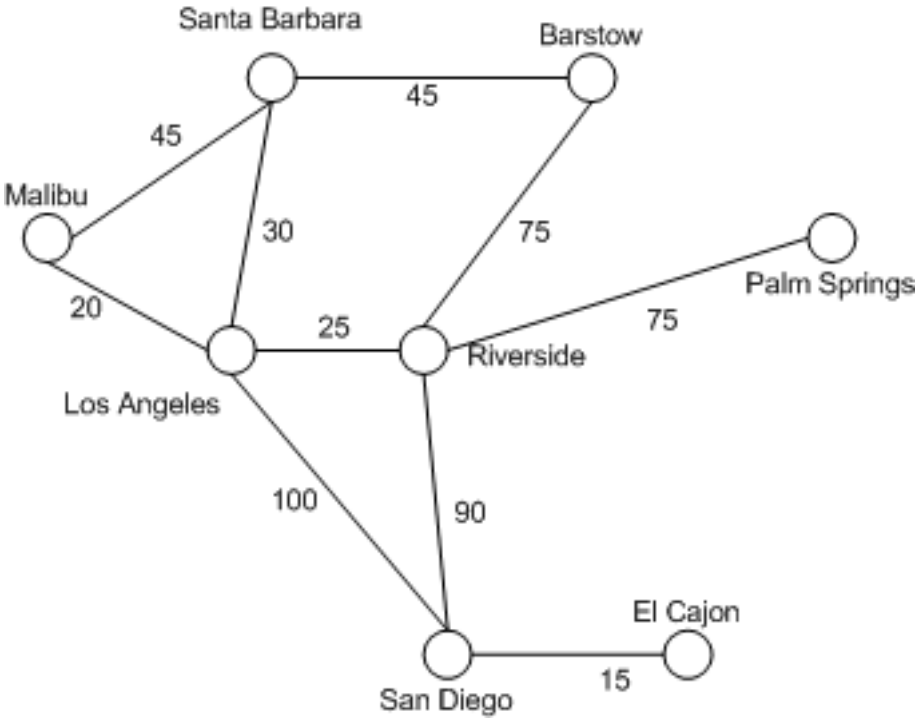

Figura 6. Grafo de ciudades de California con aristas evaluadas en millas.

Notar que la direccionalidad y el peso son ortogonales, es decir, que un grafo puede tener una de las cuatro disposiciones de aristas:

Dirigidas con peso.

Dirigidas sin peso.

No dirigidas con peso.

No dirigidas con peso.

Los grafos de la figura 4 son no dirigidos y sin peso, el de la figura 5 es dirigido y sin peso y el de la figura 6 es no dirigido y con peso.

- **Nodos**

La clase *Node* representa un nodo simple en el grafo. Cuando se trabaja con grafos, los nodos son típicamente representan una entidad. Normalmente tienen como propiedades una clave (Key) que lo identifique únicamente, y un valor (data), donde se almacena información acerca del nodo.

- **Listas de adyacencia**

Es una colección de objetos que hacen referencia a otros. En este caso se usa para almacenar las aristas que tiene cada nodo hacia sus vecinos.

Un inconveniente de la lista adyacente es la determinación de que si una arista de un nodo u a v requiere ser buscada en la lista adyacente de u. Para gráficos densos u probablemente tendrá muchos nodos en su lista de adyacencia. La determinación de si existe una arista entre dos nodos se distribuye linealmente con el tiempo para grafos con listas de adyacencia densas.

- **Tabla Hash (Hashtable)**

Representa una colección de pares de clave y valor organizados en función del código hash de la clave. Cada elemento es un par de clave y valor almacenado en un objeto *DictionaryEntry*. Una clave no puede ser referencia de objeto *null*, pero un valor sí puede serlo.

Los objetos de claves deben permanecer inmutables mientras se utilicen como claves en *Hashtable*.

Cuando se agrega un elemento a *Hashtable*, el elemento se coloca en un sector de almacenamiento en función del código hash de la clave. Las búsquedas posteriores de la clave utilizarán su código hash para buscar en un sector de almacenamiento determinado solamente; de este modo, se reducirá considerablemente el número de comparaciones de clave necesarias para encontrar un elemento.

La capacidad de *Hashtable* es el número de elementos que puede contener. La capacidad inicial predeterminada de es cero. Conforme se agregan elementos a *Hashtable*, la capacidad aumenta automáticamente según lo requiera la reasignación.

La instrucción *foreach* del lenguaje C# requiere el tipo de cada elemento de la colección. Como los elementos de son pares de clave y valor, el tipo del elemento no se corresponde con el tipo de la clave ni con el del valor.

Con esta estructura de datos se obtienen las colecciones de rutas y distancias de todos los nodos del grafo, para poder acceder a los nodos de una forma sencilla mediante el conjunto clave y valor, donde la clave será el nombre del nodo y el valor un objeto coordenadas donde se almacenan las posiciones de los nodos dentro del entorno simulado.

2.3.2. Técnicas para generación de trayectorias.

Las técnicas de planificación de trayectorias tienen como objetivo el cálculo de los movimientos que necesita realizar el robot para llegar a un punto determinado desde su posición actual. A continuación se describen brevemente los puntos más relevantes relativos a estas técnicas.

- **Planificación de caminos (*Path Planning*) Local y Global.**

Se define navegación como la metodología (o arte) que permite guiar el curso de un robot móvil a través de un entorno con obstáculos. Existen diversos esquemas, pero todos ellos poseen en común el afán por llevar el vehículo a su

destino de forma segura. La capacidad de reacción ante situaciones inesperadas debe ser la principal cualidad para desenvolverse, de modo eficaz, en entornos no estructurados. Las tareas involucradas en la navegación de un robot móvil son: la percepción del entorno a través de sus sensores, de modo que le permita crear una abstracción del mundo; la planificación de una trayectoria libre de obstáculos, para alcanzar el punto destino seleccionado; y el guiado del vehículo a través de la referencia construida. De forma simultánea, el vehículo puede interaccionar con ciertos elementos del entorno. Así, se define el concepto de operación como la programación de las herramientas de a bordo que le permiten realizar la tarea especificada.

Realizar una tarea de navegación para un robot móvil significa recorrer un camino que lo conduzca desde una posición inicial hasta otra final, pasando por ciertas posiciones intermedias o submetas. El problema de la navegación se divide en las siguientes cuatro etapas:

- Percepción del mundo: Mediante el uso de sensores externos, creación de un mapa o modelo del entorno donde se desarrollará la tarea de navegación. [*7*]

- Planificación de la ruta: Crea una secuencia ordenada de objetivos o submetas que deben ser alcanzadas por el vehículo. Esta secuencia se calcula utilizando el modelo o mapa de entorno, la descripción de la tarea que debe realizar y algún tipo de procedimiento estratégico.

- Generación del camino: En primer lugar define una función contínua que interpola la secuencia de objetivos construida por el planificador. Posteriormente procede a la discretización de la misma a fin de generar el camino.

- Seguimiento del camino: Efectúa el desplazamiento del vehículo, según el camino generado mediante el adecuado control de los actuadores del vehículo. [*8*]

Estas tareas pueden llevarse a cabo de forma separada, aunque en el orden especificado. La interrelación existente entre cada una de estas tareas conforma la estructura de control de navegación básica en un robot móvil [9] y se muestra en la siguiente figura.

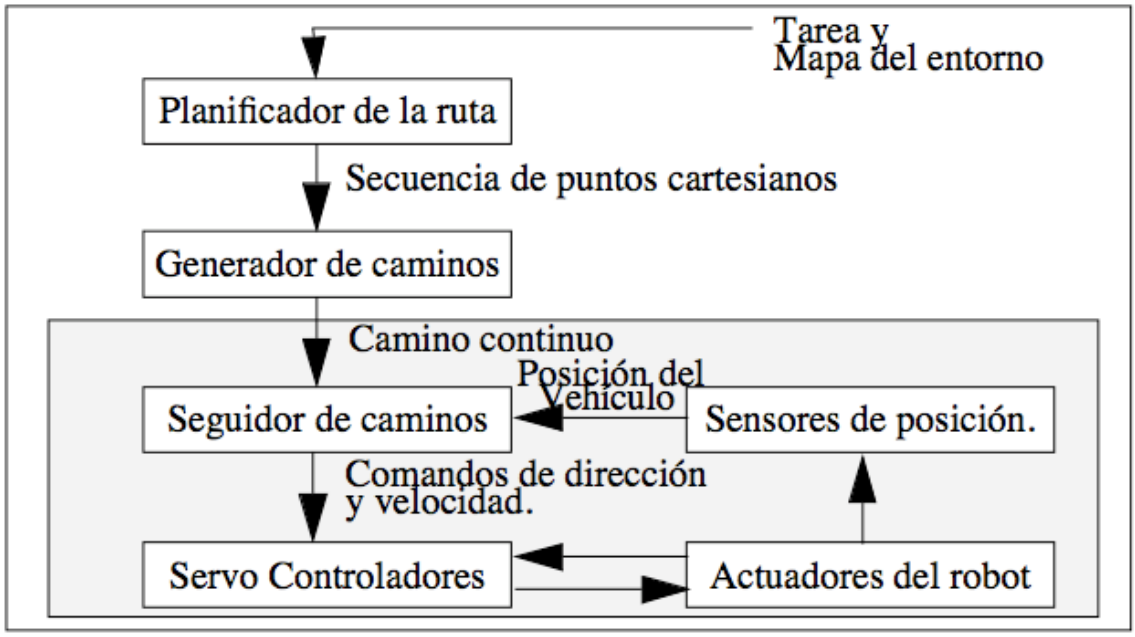

Figura 7. Estructura de navegación básica en un robot móvil

En el esquema dispuesto en la figura 7, se parte de un mapa de entorno y de las especificaciones de la tarea de navegación. De estos datos se realiza la planificación de un conjunto de objetivos representados como una secuencia de puntos cartesianos dispersos que definen la ruta. Dicho conjunto cumple los requisitos de la tarea impuesta asegurándose de que la ruta asociada está libre de obstáculos. Mediante el uso del generador del camino se construye la referencia que utilizará el seguidor de caminos para generar los comandos de direccionamiento y velocidad que actuarán sobre los servo controladores del vehículo. Por último, mediante el uso de los sensores internos del vehículo (sensores de posición) en conjunción con técnicas odométricas, se produce una estimación de la posición actual [10], la cual será realimentada al seguidor de caminos.

La complejidad del sistema necesario para desarrollar esta tarea depende principalmente del conocimiento que se posee del entorno de trabajo. Así, la figura 7 considera que se cuenta con un mapa del entorno que responde de forma fiel a la realidad.

Mediante el uso adecuado del mismo se puede construir un camino que cumpla los requisitos impuestos por la tarea de navegación, sin que el vehículo colisione con algún elemento del entorno.

Sin embargo, es posible que el modelo del entorno del que dispone el robot adolezca de ciertas imperfecciones al omitir algunos detalles del mismo. El esquema presentado en la figura 7 resulta ineficaz, al no asegurar la construcción de un camino libre de obstáculos. Por ello se necesita introducir en la estructura de control básica nuevos elementos que palien este defecto.

Un esquema de navegador utilizado en aplicaciones, donde la información acerca del entorno de trabajo varía desde un perfecto conocimiento del mismo hasta poseer un cierto grado de incertidumbre [11], es el mostrado en la figura 8:

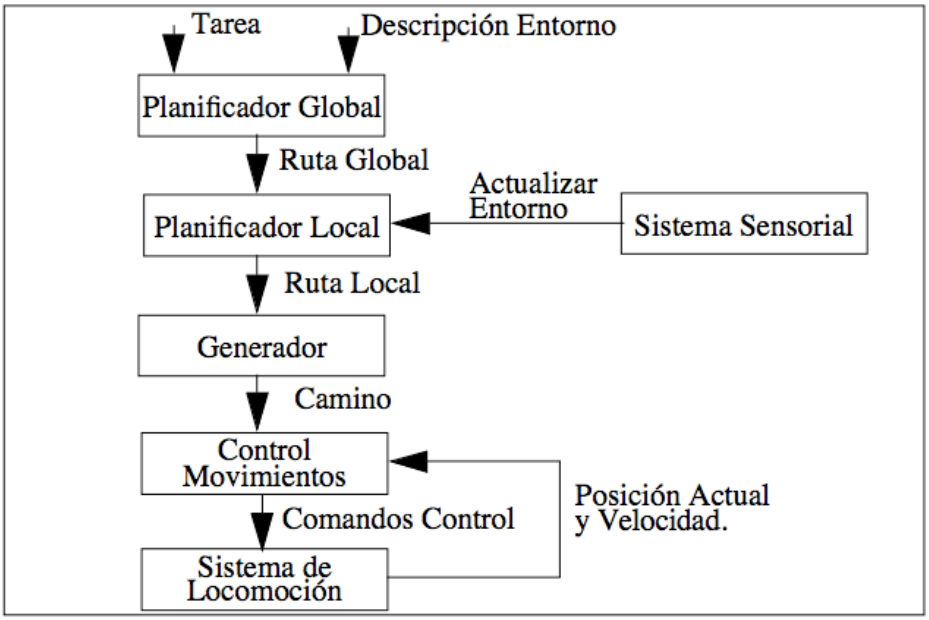

Figura 8. Estructura de navegación básica

El esquema presentado contiene en líneas generales, el funcionamiento de la estructura de navegación básica. Lo novedoso reside en el desdoblamiento de la tarea de planificación en dos subtareas: planificación global y local. La primera de estas subtareas es análoga al módulo de planificación de la figura 7 y construye una ruta sobre la cual se puede definir un camino libre de obstáculos según la información que a priori se posee del entorno. Si la descripción del entorno introducida fuese perfecta, la ruta calculada sería de forma directa la entrada de la tarea generador. Sin embargo, al no serlo, puede dar lugar a la construcción de un camino que no esté libre de obstáculos, con el consiguiente peligro de que el vehículo impacte con algún elemento del entorno. La tarea de planificación local recibe información del sistema sensorial sobre el entorno local del robot, según el radio de alcance de los sensores externos de a bordo. Mediante el análisis de estos datos actualiza el modelo preliminar del entorno y decide si se precisa replanificar la ruta local del robot.

La clave del esquema presentado en la figura 8 para adaptarse a diversos entornos, aunque no se posea un conocimiento exhaustivo del mismo, reside en la distinción efectuada entre planificación global y local. Ambos conceptos se pueden definir con mayor precisión de forma que sigue [12]:

- **Planificación global.**

Construir o planificar la ruta que lleve al robot a cada una de las submetas determinadas por el control de misión, según las especificaciones del problema que debe resolverse. Esta planificación es una aproximación al camino final que se va a seguir, ya que en la realización de esta acción no se consideran los detalles del entorno local al vehículo.

- **Planificación local**

Resolver las obstrucciones sobre la ruta global en el entorno local al robot para determinar la ruta real que será seguida. El modelo del entorno local se construye mediante la fusión de la información proporcionada por los sensores externos del robot móvil.

La construcción de la ruta global puede realizarse antes de que el vehículo comience a ejecutar la tarea, mientras que la planificación local se lleva a cabo en tiempo de ejecución. En el caso de realizar una navegación sobre entornos totalmente conocidos es obvio que resulta innecesario proceder a una planificación local, pero a medida que disminuye el conocimiento de la zona por la cual el robot móvil realiza su tarea, aumenta la relevancia de la misma.

2.3.3. Slam

Dentro de las muchas labores que afrontan los robots, se ven enfrentados a ambientes dinámicos en el cual las recientes aplicaciones de navegación e interacción con otros robots y con seres humanos, requieren tener bajo control su desplazamiento, evitando colisiones con elementos estáticos o dinámicos de su entorno. Por esto, a lo largo de los últimos años la investigación en el campo de la robótica ha abordado el problema de mapeo y localización simultánea en dichos ambientes, lo cual se conoce como SLAM (Simultaneous Localization And Maping) que en su modalidad visual involucra diversas áreas como son: la visión por computador, el procesamiento de señales, la inteligencia artificial y es actualmente un tema abierto de investigación.

La problemática que encierra el desplazamiento de los robots en escenas dinámicas y complejas se puede clasificar en tres grandes campos a saber:

- reconstrucción de la parte estática de la escena, llamada “mundo”
- conocer donde se encuentra el robot dentro de este mudo y en relación a los objetos que se tienen
- detectar los objetos en movimiento, identificarlos, describir su posición y velocidad para la estimación en un tiempo futuro

Las técnicas de mapeo y localización simultánea conocidas como SLAM son actualmente investigadas por numerosos grupos y laboratorios del mundo. El SLAM puede realizarse con diferentes tipos de sensores pero en los últimos años ha tomado fuerza el uso de cámaras de video y en especial cámaras estereoscópicas dado que permiten tener información adicional que facilita el cálculo de la profundidad. La información obtenida mediante las cámaras es tratada con diferentes técnicas de procesamiento digital de imágenes con el objeto de extraer características y simplificar el problema. Básicamente se utiliza detección de bordes y el uso de descriptores. En general la información extraída es incompleta y con ruido como sucede con cualquier tipo de sensor, aquí se hace necesario el uso de filtros, entre los cuales se destacan el de Kalman y recientemente los filtros de partículas.

A menudo se busca un entendimiento de la escena en términos de modelos u objetos predefinidos en una base de conocimiento lo cual se conoce como visión basada en modelos. El entendimiento automático de una escena se ve dificultado porque la imagen capturada es una combinación de fuentes luminosas, materiales y objetos que a menudo cubren a otros, lo cual implica la utilización de técnicas de reconocimiento de patrones y computación gráfica. Para realizar el mapeo y localización en tiempo real la información extraída debe ser almacenada mediante estructuras de datos adecuadas y algoritmos eficientes.

Según [*13*], la navegación puede dividirse en los siguientes tipos:

- Navegación basada en mapas. Estos sistemas dependen de modelos geométricos creados por el usuario o mapas topológicos del ambiente.
- Navegación basada en la construcción de mapas. Estos son sistemas que usan sensores para construir sus propios modelos geométricos o topológicos del ambiente y los usan para navegación.

- Navegación sin mapas. Estos son sistemas que no usan representación explícita de todo lo que hay en el espacio de navegación si no en reconocer o reorganizar los objetos que se encuentran allí.

De acuerdo a [*14*], los cuatro pasos generales para realizar la localización de un robot son:

- Adquirir información sensorial. Para navegación visual, esto significa capturar las imágenes provenientes de las cámaras.
- Detectar marcas. Realizar el procesamiento de la imagen para detectar los puntos de interés en las imágenes capturadas.
- Establecer correspondencias entre lo observado y lo esperado. Esto asume analizar lo observado con respecto a una base de datos o unos criterios bien definidos de comparación para poder discernir qué es lo que está viendo el robot.
- Cálculo de la posición. Gracias a la comparación realizada se puede hacer el cálculo de la posición del robot.

El problema de construcción de mapas por el mismo robot es uno de los problemas fundamentales en robótica móvil. Éste involucra el resolver concurrentemente los problemas de construcción de un mapa y localizarse en él mientras se está construyendo.

3. HERRAMIENTAS UTILIZADAS

3. HERRAMIENTAS UTILIZADAS

3.1. MRDS

Microsoft Robotics Developer Studio es un entorno orientado a la creación de aplicaciones para robots con gran variedad de hardware. Dicho entorno consta de una plataforma de desarrollo de extremo a extremo, escalable y extensible que incluye lo siguiente:

1. Un lenguaje visual de programación, que permite a cualquier persona la creación fácilmente de aplicaciones para robots.

2. Un simulador de entornos en tres dimensiones para aplicaciones de robótica.

3. Soporte de tiempo de ejecución (*runtime*) ligero, que permite aplicaciones en tiempo real con gran variedad de hardware.

Será necesario el aprendizaje y manejo de este entorno para poder desarrollar todo el proyecto.

MRDS permite crear aplicaciones basadas en servicios para una amplia gama de dispositivos de hardware. El kit de herramientas incluye un módulo de tiempo de ejecución que le resultará familiar a los desarrolladores de Windows® Communication Framework (WCF). Además, incluye una herramienta de lenguaje de programación visual (VPL) y un entorno de simulación visual (VSE).

MRDS proporciona un módulo de tiempo de ejecución orientado a servicios, además de las herramientas necesarias para diseñar e implementar aplicaciones basadas en robótica. Incluye herramientas, tutoriales y documentación para crear elementos visuales, diseñados para introducir rápidamente a los diseñadores con poca experiencia en el mundo de la robótica. Los desarrolladores comerciales deberán adquirir el kit de herramientas por una pequeña cantidad, pero los

desarrolladores aficionados e investigadores académicos podrán descargarlo y usarlo de forma gratuita.

El soporte de tiempo de ejecución de MRDS consiste en dos componentes de menor nivel que se basan en CLR (Common Language Runtime) 2.0. Estos dos soportes de tiempo de ejecución son los servicios de software descentralizado (DSS) y el módulo de tiempo de ejecución de simultaneidad y coordinación (CCR). DSS es un módulo de tiempo de ejecución ligero y orientado al servicio, basado en los principios de Transferencia de estado de representación (REST) que se usan para aumentar la eficacia de la Web. CCR es una biblioteca de Microsoft .NET Framework compatible con el procesamiento asincrónico. Esto tiene una importancia vital para las aplicaciones de robótica, ya que hay numerosos sensores y actuadores que están continuamente enviando y recibiendo datos.

Además del módulo de tiempo de ejecución, el kit de herramientas de MRDS incluye una herramienta VPL que permite la creación de aplicaciones de robótica si se arrastran los elementos a una superficie de diseño. MRDS también incluye un VSE que le permite experimentar con simulaciones complejas que impliquen a varios robots y obstáculos.

Una de las ventajas de MDRS es que permite interactuar y controlar robots a través de aplicaciones Windows o aplicaciones web. Dichas aplicaciones se crean en un entorno de desarrollo llamado plataforma .NET. La plataforma .NET se puede definir como una capa que se coloca entre el sistema operativo (SO) y el programador abstrayendo los detalles internos del SO. Más concretamente, es un entorno para la construcción, desarrollo y ejecución de servicios web y otras aplicaciones que consiste en tres partes fundamentales: entorno de ejecución denominado CLR (Common Language Runtime), las clases de la plataforma (Framework Classes) y lenguajes soportados por la plataforma (ASP.NET, C#, IronPython, etc.).

Una aplicación se basa en la entrada de información a través de un sensor y la respuesta de un actuador a dicha información. Esto se realiza a través de una aplicación de MRDS, orquestación de servicios (coordinación de servicios). Es decir, el sensor es representado por un servicio, al igual que el actuador y el controlador. La coordinación se produce en la comunicación asíncrona entre todos esos servicios. Por ejemplo, si un robot consta de un parachoques, en el momento que se produzca

un choque, el sensor detecta el contacto con el objeto y manda un mensaje al controlador que decidirá que mensaje enviará al actuador para realizar la operación oportuna.

Entrando en detalle, dentro de la estructura de MSRS está el *Runtime*, componente capaz de soportar una gran variedad de hardware dentro de robots conectados directamente a un ordenador personal o robots simulados que pueden ser manipulados, como ellos funcionan, en un mundo virtual. Además el *Runtime* está adaptado para soportar cualquier tipo de aplicación como sensores de entrada, conducción por cable o remota, autonomía del propio robot o cooperaciones entre robots autónomos.

El CCR permite la coordinación de mensajes, abstrayendo al programador del uso de hilos, semáforos, etc. elementos de programación que coordinan el flujo de información. Además plantea un modelo de programa que facilita las operaciones asíncronas explotando el hardware paralelo. Cabe destacar que CCR es un componente DLL (*Dynamic Linking Library***)** autónomo de .NET accesible desde cualquier lenguaje de programación abarcado por .NET.

El DSS combina la arquitectura tradicional Web con pedazos de arquitectura de Servicios Web. La arquitectura resultante se basa en servicios para coordinar las aplicaciones distribuidas. Una aplicación es, por tanto, un conjunto de servicios que se coordinan entre sí.

El principal objetivo es promover simplicidad e interoperabilidad, esto se consigue creando usos como las composiciones de servicios sin importar si estos servicios están funcionando dentro del mismo nodo o a través de la red. El resultado es una plataforma altamente flexible capaz de escribir un amplio sistema de aplicaciones. El DSS utiliza los protocolos HTTP y DSSP (*Descentralized Software Services Protocol*).

Hay que decir que DSSP es un protocolo propio que ofrece DSS encargado de la mensajería entre servicios y la creación de servicios descentralizados. Además permite que el estado se mantenga durante el periodo de vida de la aplicación.

En cuanto a los nodos DSS, son los encargados de coordinar las actividades de todos los servicios.

Un aspecto muy importante es llegar a controlar múltiples tareas interactivas simultáneas y conseguir que se realice en tiempos diferentes. Esto se conoce como programación asíncrona y concurrente. La herramienta MSRS consigue esta capacidad a través de una parte del *Runtime* comentada anteriormente, el CCR. Además el CCR ofrece un gran número de operaciones primitivas que permiten al programador expresar fácilmente este modelo de coordinación tan complejo.

3.2. Herramienta de desarrollo

Para la consecución del objetivo, es imprescindible analizar la herramienta de desarrollo con la que se creará la aplicación. Se estudiarán dos posibilidades:

3.2.1. Visual Programming Language

Microsoft Visual Programming Language (VPL) es un entorno de desarrollo de aplicaciones basado en la definición de flujos de forma gráfica. A diferencia de series o comandos imperativos ejecutados secuencialmente, un programa de flujo de datos es como una serie de trabajadores en una línea de ensamblaje, los cuales hacen sus tareas en cuanto llegan los materiales. Como resultado VPL se adapta bien a una variación de la programación concurrente o de escenarios de procedimientos distribuidos.

VPL está pensado para empezar a programar con un conocimiento básico de los conceptos fundamentales, como los de variable y lógica. Sin embargo, VPL no se limita a los principiantes. Este lenguaje de programación puede ser también de interés para los programadores más avanzados para el rápido desarrollo de prototipos o código. Además, aunque su caja de herramientas se adapta al desarrollo de aplicaciones para robots, la arquitectura subyacente no está limitada para la programación de robots y podría aplicarse también para otro tipo de aplicaciones. Como resultado VPL abarca a una amplia audiencia de usuarios, entre ellos los estudiantes, entusiastas y aficionados, así como posiblemente, a los desarrolladores web y programadores profesionales.

El flujo de datos de Microsoft Visual Programming Lenguaje consiste en una secuencia de conexiones representadas por bloques con entradas y salidas que pueden conectarse a otros bloques de actividad.

Las actividades pueden representar definiciones de servicios, el control de flujos de datos, funciones u otros módulos de código. La aplicación resultante es por lo tanto, a menudo denominada orquestación (coordinación).

Las actividades pueden incluirse asimismo como composiciones de otras actividades. Éstas hacen posible la composición de actividades en base a la reutilización de bloques funcionales. En este sentido, una aplicación construida en VPL es en sí misma una actividad.

La implementación de una aplicación en VPL es bastante sencilla. Para comprobarlo, se han implementado los tutoriales de VPL donde se describe los pasos para crear pequeños ejemplos, como puede ser el “hola mundo” o un contador. El problema que radica de la utilización de esta herramienta para personas que tienen conocimientos de programación es que resulta poco intuitivo y bastante complicado encontrar errores de ejecución.

Por este motivo se ha preferido no usar la herramienta VPL para el desarrollo de la aplicación, porque resulta más complejo desde un punto de vista personal, aunque el manejo de la herramienta sea sencillo.

Principalmente VPL está diseñado para personas que se interesan por el mundo de la robótica pero que carecen de conocimientos extendidos en programación o para especificar la arquitectura inicial de la aplicación.

3.2.2. Visual C#

Microsoft Visual C# 2008 es un lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. C# es simple, eficaz, con seguridad de tipos y orientado a objetos. Con sus diversas

innovaciones, C# permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

Visual Studio admite Visual C# con un editor de código completo, plantillas de proyecto, diseñadores, asistentes para código, un depurador eficaz y fácil de usar, además de otras herramientas. La biblioteca de clases .NET Framework ofrece acceso a una amplia gama de servicios de sistema operativo y a otras clases útiles y adecuadamente diseñadas que aceleran el ciclo de desarrollo de manera significativa.

C# es un lenguaje con seguridad en el tratamiento de tipos, que es a la vez sencillo y potente y permite a los programadores crear una gran variedad de aplicaciones. Combinado con .NET Framework, Visual C# permite la creación de aplicaciones para Windows, servicios Web, herramientas de base de datos, componentes, controles y mucho más.

En este caso la arquitectura inicial ya está creada, por eso se prescinde del VPL para ello y dado que para una persona que haya trabajado con diferentes lenguajes de programación, manejar uno más es relativamente sencillo. Por tanto se partirá del código del ejemplo *Follower* proporcionado por MRDS y se ampliará hasta la consecución del objetivo propuesto utilizando C# como herramienta de desarrollo de la aplicación.

3.3. VSE (Visual Simulation Environment)

Se generará el entorno virtual con el que se construirá la estructura que representará el hotel, con habitaciones y todo su mobiliario, la recepción del hotel, cafetería y demás elementos con los que se va a trabajar. Se utilizará la herramienta proporcionada con MRDS.

3.4. SolidWorks

SolidWorks es una herramienta de diseño en 3D. El software de CAD de SolidWorks permite realizar diseños en tres dimensiones. Todos los productos SolidWorks están completamente integrados y se pueden combinar. Esto significa que es posible trabajar en una sola ventana y los cambios se actualizarán de forma automática en el resto de aplicaciones integradas.

3.5. Integración de servicios

MRDS es una plataforma orientada a servicios que incluye servicios en tiempo de ejecución basados en .NET.

Por otro lado, soporta la monitorización del robot en tiempo real de sensores y respuesta de motores y actuadores. El DSS (Descentralized Software Services) da soporte a servicios distribuidos, combinando la arquitectura tradicional Web con pedazos de arquitectura de servicios Web. Típicamente una aplicación corre en un nodo DSS, que son los encargados de coordinar las actividades de todos los servicios. La aplicación del modelo del software de servicios descentralizados, hace que sea sencillo el acceso, y la respuesta del estado del robot, usando un navegador Web o una aplicación basada en Windows.

También facilita la reutilización modular de los servicios usando un modelo compuesto. Es posible construir funciones de alto nivel usando componentes simples que prevé la reutilización del código de los módulos con una mayor seguridad e integridad.

En este proyecto se integran se integran varios tipos de servicios que se analizarán posteriormente.

4. DISEÑO DE LA SOLUCIÓN

4. DISEÑO DE LA SOLUCIÓN

A continuación se describen los servicios software implementados como parte del desarrollo de la aplicación de robot guía y cómo éstos se estructuran en una aplicación robótica.

4.1. Arquitectura de la Aplicación

La aplicación diseñada se basa en una arquitectura orientada a servicios. Cada componente básico de la aplicación es un servicio DSS de Robotics Developer Studio. La comunicación entre servicios se realiza mediante el protocolo DSSP.

DSSP es un protocolo simple basado en SOAP (Simple Object Access Protocol) que define un modelo del servicio con las nociones de identidad, estado y enlaces entre los servicios. DSSP define un conjunto de mensajes de estado (también denominados operaciones) que proporcionan la infraestructura necesaria para la extracción de datos estructurados, su manipulación y la notificación de eventos. El objetivo de DSSP es permitir la definición de aplicaciones como composiciones de los servicios que funcionan colaborativamente en un ambiente descentralizado. Un aspecto a tener en cuenta es que cada servicio puede albergarse en una máquina distinta, hecho que proporciona una potencia enorme en la creación de aplicaciones.

La funcionalidad proporcionada por DSSP es una extensión del modelo de la aplicación proporcionado por el HTTP y es utilizada como ampliación a la infraestructura existente del http.

DSSP define una aplicación como composición de los servicios que se pueden aprovechar para alcanzar una tarea deseada con la orquestación (coordinación). Los servicios son las entidades que pueden ser creadas, manipuladas, vigiladas, y destruidas en varias ocasiones sobre el curso de la vida de una aplicación usando las operaciones definidas por DSSP.

Un servicio consiste en:

- *Identity*: referencia única del servicio.
- *Behavior*: la definición de las funciones del servicio.
- *Service State*: el estado del servicio
- *Service Context*: los lazos que el servicio tiene con otros servicios.

DSSP proporciona a un modelo uniforme para crear, suprimir, manipular, suscribir, y orquestar independientemente de los servicios de la semántica de los mismos. Permite pero no requiere un modelo de datos compartidos a través de todos los servicios. Como consecuencia, algunos de los mensajes de DSSP son concretos y otros son polimórficos y se deben adaptar al modelo contento determinado de un servicio.

Un servicio de DSSP tiene un identificador del servicio global y único. El identificador del servicio proporciona solamente su identidad; no transporta ninguna información sobre el estado, el comportamiento, o el contexto del servicio.

El estado del servicio es una representación de un servicio en todo momento. Cualquier información que deba ser extraída usando DSSP se debe expresar como parte del estado del servicio.

El comportamiento de un servicio (el contrato) es la combinación del modelo que describe el estado y los intercambios del mensaje que un servicio define para comunicar con otros servicios.

Una aplicación es una composición de los servicios que, con la orquestación, se pueden aprovechar para alcanzar una tarea deseada. Existen mecanismos proporcionados para crear y destruir servicios, para extraer y manipular su estado, y para suscribir para las notificaciones del acontecimiento como resultado de cambios de estado.

DSSP define un evento como cambio de estado en un servicio. Por ejemplo, en el caso de una operación de actualización en un servicio, el estado de ese servicio cambia como un resultado directo de esa operación de actualización. Estos

cambios de estado se envían mediante notificaciones. Un servicio puede recibir notificaciones de un acontecimiento suscribiéndose a un servicio.

Cuando un servicio suscribe a un editor, cualquier emparejamiento existente en la suscripción se envía notificaciones al suscriptor, que permite conocer el estado actual del editor.

Los servicios son un bloque básico para construir aplicaciones en MRSR y es un componente básico en el modelo de la aplicación. El modelo de servicios DSSP ha sido diseñado para facilitar el reutilización de los servicios, consiguiendo un manejo sencillo.

Una vez explicadas las bases de la arquitectura de la aplicación creada, el siguiente paso es explicar qué relación tienen los servicios utilizados en el desarrollo. Primero, se muestra la figura representativa de la arquitectura de la aplicación:

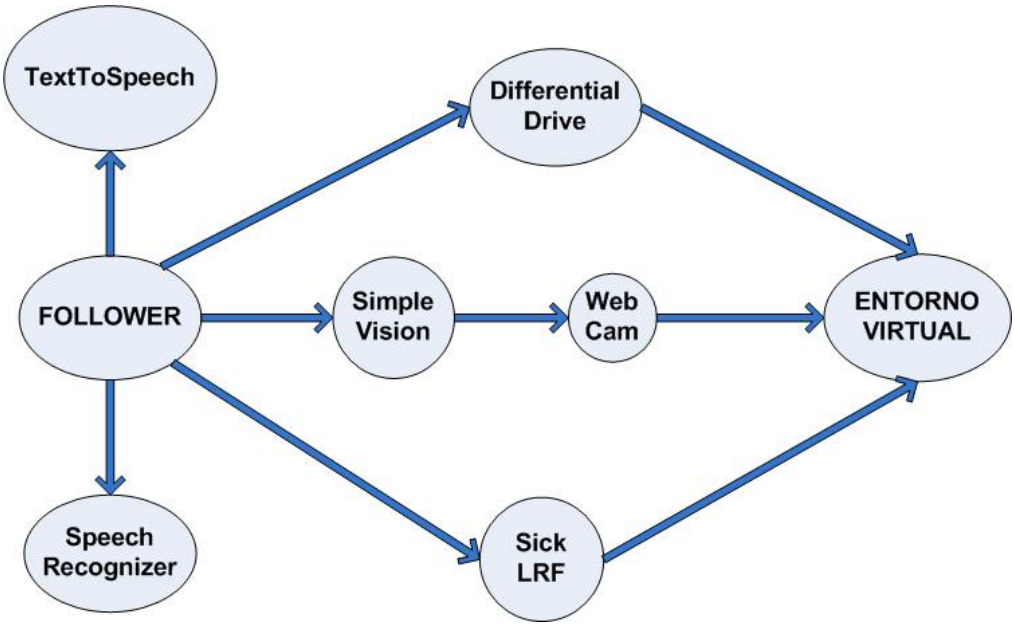

Figura 9: Arquitectura de la aplicación

El servicio principal es el *Follower*, que soporta la mayor funcionalidad. Éste hace de socio (*partner*) del servicio *TextToSpeech* para poder enviarle mensajes

(operaciones) con el texto para que éste sintetice la voz. El servicio *Follower* suscribe a los siguientes servicios:

- SickLRF: para recibir notificaciones del láser
- SimpleVision: para obtener información de la WebCam, a su vez, servicio que suscribe SimpleVision para obtener la información del entorno simulado.
- DifferentialDrive: para enviar y recibir notificaciones del estado del motor y con el que se realiza el movimiento por el entorno simulado.
- SpeechRecognizer: para recibir notificaciones de voz por parte del robot (este servicio no se utiliza en la aplicación pero está preparada para soportarlo).

Una vez visualizado el esquema de la arquitectura de la aplicación, se explicará en los siguientes puntos las características y funcionalidad de los servicios expuestos.

4.2. Entorno de simulación

Uno de los servicios de MRDS es la recreación de un ambiente virtual parecido a un laberinto y el experimento con un robot Pioneer 3 DX simulado equipado con una serie de transductores sónar frontales y con una cámara para poder reconocer caras y objetos.

Uno de los mayores motivos por el que se creó MRDS era acelerar el desarrollo y adopción de la robótica. Gran parte de este esfuerzo está centrado en la simulación. Se ha diseñado una simulación con una alta fidelidad, escalabilidad y visualización, que permite a cualquier persona usarlo como si fuera un juego.

Pero antes de todo, ¿qué es una simulación? Imaginar un folioscopio, un libro pequeño con una imagen en cada página que "se mueve" cuando se pasan rápidamente las páginas. Las simulaciones funcionan de un modo semejante. Una

simulación contiene una o más entidades. Cada una de ellas se presenta en un fotograma. Cada fotograma equivale a una página del folioscopio. La velocidad a la que se presentan los fotogramas depende de la tarjeta gráfica del equipo host.

La diferencia principal entre el folioscopio y la simulación es que el primero es estático o predecible, mientras que una simulación no lo es. Cada fotograma de la simulación se presenta de forma dinámica y las fuerzas con las que se encuentra una entidad en cada fotograma no se conocen siempre de antemano. Esto es lo que convierte a las simulaciones en algo tan útil. Puede representar un escenario con un robot y observar cómo interactúa éste con el mundo.

La simulación permite a la gente con un ordenador personal desarrollar robots muy interesantes. Al mismo tiempo, pueden probar los resultados en robots físicos, es decir, podrán ver sus esfuerzos en algo realizable. Además, permite al programador eliminar fallos básicos que se podrán probar con antelación en el entorno virtual.

MRDS usa el motor de AGEIA PhysX para proporcionar las reglas físicas del entorno de simulación. Sin la física, la simulación no tendría utilidad alguna, ya que no sería un reflejo del mundo que intenta simular. Por ejemplo, sin gravedad las entidades podrían flotar.

Otra característica, es que se puede simular al mismo tiempo que es robots está ejecutándose. Pero la simulación tiene sus límites, ya que el mundo real es complejo y está lleno de imprevistos y de ruido.

La simulación está compuesta de varios componentes:

- Simulation Engine Service: responsable del progreso del tiempo en el motor físico de la simulación.

- Managed Physics Engine Wrapper: abstrae al programador del nivel bajo del motor físico.

- Native Physics Engine Library: permite la aceleración del hardware a través de *AGEIA PhysX Technology.*

- Entities**:** representa el hardware y los objetos físicos en el mundo simulado. Un gran número de entidades están definidas en MRDS y permiten crear rápidamente un entorno de simulación.

Cada elemento del hotel se encuentra dentro de este grupo. Se han creado camas, almohadas, mesitas de noche, lámparas, sanitarios, sofás, mesas y sillas. En realidad, la escena de simulación también incluye las entidades que representan la cámara principal, el suelo, el cielo y el sol.

El modo de edición, tal y como se muestra en la figura 10, incluye un panel a la izquierda en el que puede modificar las propiedades asociadas a cada entidad. Estas propiedades lo controlan todo, desde el nombre de la entidad hasta su posición en el entorno de la simulación. Además, también le permiten controlar con precisión el modo en que se presenta la entidad, lo que afecta a cómo aparece en la simulación.

Figura 10. Modo de edición para modificar las propiedades de la entidad

Si se vuelve al modo de ejecución, se puede mover por la simulación con el ratón o las teclas de dirección. De este modo se cambia el punto de vista de la cámara principal, que es su propio punto de vista en la simulación. También es

importante mencionar que VSE (entorno de simulación virtual de Microsoft) permite presentar la escena de diferentes modos. El modo visual es el modo predeterminado y consiste en una visión realista de la escena de simulación. Sin embargo, el elemento del menú Render permite elegir entre los modos Wireframe, Physics, Combined o No Rendering.

La opción No Rendering se incluye porque la representación sólo es un aspecto más de la simulación. Lo más valioso de una simulación es la interacción entre diversas entidades. Como la representación de entidades en una escena de simulación es bastante costosa en cuanto a recursos, la opción No Rendering puede ser bastante útil cuando haya un gran número de entidades implicadas.

4.2.1. Entidades

Un tipo de entidad permite definir una nueva instancia de un tipo de entidad determinado. Por ejemplo, el globo terráqueo incluido en el entorno básico de simulación es un tipo de entidad de forma única. El tipo de entidad actúa como una plantilla para la nueva entidad, en el sentido de que especifica las propiedades asociadas a un tipo de entidad determinada. Una vez que se haya creado la entidad, es posible modificar los valores de estas propiedades, pero el tipo de entidad es el que define las propiedades que se incluyen.

VSE precisa un tipo de entidad para agregar una entidad a una simulación. Por ejemplo, para agregar una entidad tipo mesa, habría que agregar una nueva entidad desde VSE; para ello, se hace clic en Entity y Nuevo mientras se encuentre en el modo de edición. De este modo se abrirá el cuadro de diálogo *New Entity* (consultar la figura 11).

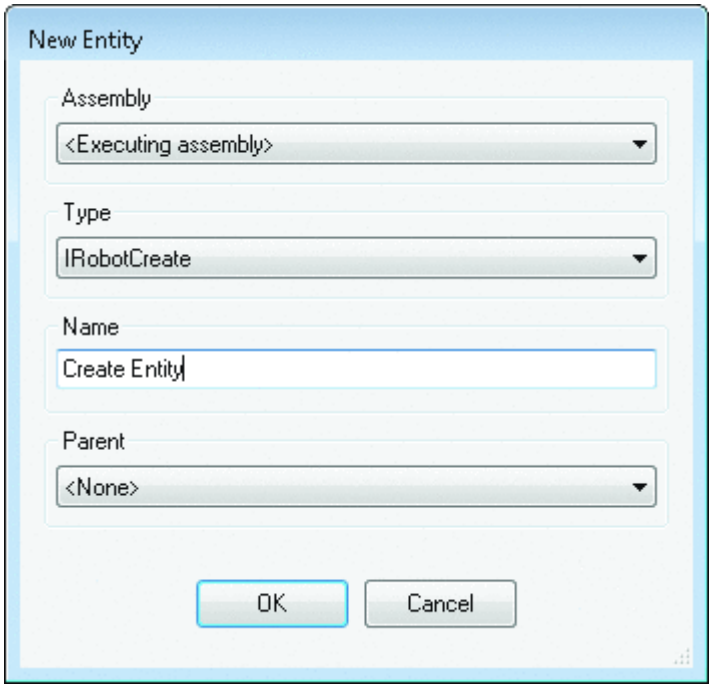

Figura 11. Inserción de una nueva entidad en el cuadro de diálogo New Entity

La malla asignada a una entidad es un archivo de objeto, que tiene una extensión de archivo .obj. El archivo de objeto se crea con una herramienta de edición gráfica en 3D y se exporta en un formato de objeto compatible. MRDS precisa que el archivo de objeto de malla tenga este formato. Lo último que debe hacer en el método *AddTexturedSphere* es insertar la entidad esférica en el entorno de la simulación.

4.2.2. Creación de una malla

Cada entidad se puede asociar a una malla tridimensional, que es lo que hace que una entidad parezca realista. Por ejemplo, en el caso del sofá, la malla es lo que permite que la entidad del sofá se parezca físicamente a un sofá, con aristas y formas. En un sentido estricto, no hace falta asociar una entidad a una malla, pero

en el caso de las entidades complejas, como es el caso de los robots, es preferible usar un objeto de malla.

Para crear una malla, se pueden usar casi todas las herramientas de edición de gráficos en 3D. Los archivos asociados a las mallas deben copiar en el directorio /store/media que está asociado a la instalación local de MRDS.

SoftImage puede exportar la imagen al formato .obj. Esto es algo que no pueden hacer todos los paquetes. Por ejemplo, SolidWorks [*15*] es un paquete 3D que recomendó MSDN en la wiki Channel9 [*16*]. Lamentablemente, SolidWorks no puede exportar al formato .obj, por lo que es recomendable una herramienta llamada Blender [*17*] para realizar la conversión.

4.2.3. Modelos 3D Simulados

Tal y como se ha explicado, con SolidWorks se ha realizado el diseño de todos los elementos del mobiliario, que son: almohadas, lavabo, váter, bañera, sofás y sillas. Para poder utilizar las piezas en MRDS, es necesario cambiarlas de formato, ya que con SolidWorks se obtienen los archivos .wrl y para esta aplicación los necesitamos en .obj, por lo que es necesaria su conversión con el programa Blender.

- **SolidWorks → Blender → MSRS**

SolidWorks es una forma fácil de usar e paquete CAD 3D. Permite un modelado rápido y preciso del robot. Como no exporta al formato .obj necesario en MSRS, se tiene que usar Blender para convertir.

Blender [*18*] es un programa de modelado en 3D fuente gratuito. Complejos modelos en SolidWorks son típicamente la unión hecha de un número de partes. Si se quiere tener realmente un buen modelo, con Blender, se pueden obtener excelentes resultados fotorealistas.

Simplemente se tiene que importar el archivo guardado en formato .wrl proporcionado por SolidWorks, guardarlo en formato .blend y posteriormente importarlo como un archivo tipo *Scene* con formato .obj. En la conversión se genera también un archivo .mtl también necesario para MRDS. Tras varias pruebas de exportación, se ha llegado a una solución que permite una recreación con total detalle de las piezas diseñadas en SolidWorks.

4.3. INTEGRACIÓN DE LOS SERVICIOS SOFTWARE

El ejemplo utilizado para el desarrollo de la aplicación dentro del paquete Microsoft Robotics Developer Studio 2008 es el *Follower*, que consta de los siguientes servicios:

4.3.1. Follower

El servicio simple *Follower* muestra cómo escribir una orquestación (coordinación) de servicios de un robot móvil para que pueda seguir a un usuario que lleve una camiseta de color. El robot está equipado con una cámara Web y un telémetro láser.

Éste ejemplo demuestra cómo usar servicios múltiples asociados como son el *SimpleVision, TextToSpeech y SpeechRecogizer* para implementar una interacción robot humana.

Este ejemplo es proporcionado por MRDS en lenguaje C# y será modificado para adecuarse a las necesidades propuestas. Gráficamente tiene el siguiente aspecto:

Figura 12. Follower inicial

El ejemplo del *Follower* está diseñado para usarlo con un robot móvil que tenga dos ruedas diferenciales, mecanismo impulsor de resbalones, sensores de contacto en las partes trasera y delantera, un dispositivo láser delantero con un haz de 180 grados y una cámara Web. Este simple servicio puede ser usado con un robot real o simulado. Los necesarios archivos manifiesto (.manifest.xml) son proporcionados en el paquete.

Si en la plataforma usada para el robot el archivo manifiesto no está disponible, se puede cambiar el manifiesto adaptándolo a la plataforma.

El servicio *Follower* proporciona una vía de interacción humana-robot. Implementa comportamientos básicos para el robot como el aviso de un obstáculo, la trayectoria, habla y gestos basados en una interacción humana.

La coordinación del servicio *Follower* tiene varios servicios asociados:

- **Drive:** usado para controlar los movimientos del robot.
- **Contact Sensor:** para evitar obstáculos.
- **Sick- LRF:** para evitar obstáculos y navegación simple.
- **SimpleVision:** detecta los colores de los objetos, regiones faciales y gestos con la mano. Asocia el color a la camiseta de la persona.
- **TextToSpeech:** para generar el habla.

- **SpeechRecognizer:** reconocer comandos de voz.

En este servicio hay dos tipos de estados del robot. Uno es el estado *lógico* que es usado para un movimiento básico. El otro es el estado de *conducta*, para el comportamiento de alto nivel del robot, que es el resultado de los comandos de voz.

Los estados lógicos del estado de conducta son:

- **Stop:** para de moverse.
- **Move:** movimiento hacia delante.
- **Turn:** giro en los grados especificados
- **Translate:** movimiento hacia delante la distancia especificada
- **Adjusted move:** movimiento hacia delante en la dirección especificada.

El estado de comportamiento del robot puede ser controlado con los siguientes comandos de voz:

- **Robot:** obtiene atención, empieza a escuchar.
- **Turn Left:** gira a la izquierda 45º.
- **Turn rigth:** gira a la derecha 45º.
- **Turn around:** gira 180º.
- **Go Foward:** empieza a avanzar. evitando obstáculos.
- **Backup:** se mueve hacia atrás.
- **Stop:** para el actual comportamiento.

Por ejemplo, “Robot, go Foward!” provoca que el robot se mueva hacia delante.

El comportamiento solo puede ser controlado usando una combinación de comandos de voz y visión basada en el reconocimiento de objetos y gestos:

- **Go there:** gira después del reconocimiento de un gesto puntual del usuario.
- **Follow me:** empieza a buscar y rastrear el color especificado (por ejemplo la camiseta del usuario).
- Algún comportamiento es provocado autónomamente sin la interacción del usuario:

- **Avoid:** evita el obstáculo usando LRF y vuelve al estado previo de comportamiento.

Alguno de estos estados de comportamiento tiene más alta prioridad que otros. Por ejemplo, el robot no ejecutará comandos de habla mientras esté en el estado *avoid*.

Tras esta descripción del servicio principal, se puede visualizar mejor la capacidad que tiene, aunque se podrá ver mejor con la simulación.

4.3.2. SickLRF

El servicio *SICK Laser Range Finder* (telémetro láser) se interconecta con la serie Sick LMS200 de telémetro láser. Éste servicio demuestra cómo interconectar el *SICK laser range finder* sobre un puerto serie. Se configura el LRF para ejecutar en un modo de monitorización continuo, ejecución de barridos de 180 grados en intervalos de 0.5 grados en el máximo de tarifa de datos disponible.

Cada vez que LRF informa de un barrido completo del servicio, el servicio aporta una notificación de reemplazo que contiene los datos LRF.

La exploración del telémetro láser proporciona una relativamente nueva y excitante alta resolución de un sensor en robótica. Comunes en la gama alta de la robótica durante muchos años, estos sensores están siendo más comunes en las aplicaciones relativamente baratas de la robótica, alta resolución y datos de alta frecuencia que generan.

Los telémetros laser pueden ser definidos como unos pequeños sonar que usan luz en lugar de sonido para crear mapas en 2D en la proximidad de los objetos cercanos.

Como el láser utiliza luz en lugar de sonido, se puede medir muy rápido y sumamente estrecho el FOV (campo visual) para la medida. El término exploración significa que se coge una lectura del rango y entonces mueve una fracción de la resolución del ángulo completo y toma otra lectura, repitiendo continuamente para

todo el círculo. Debido a la física y a limitaciones mecánicas hay a menudo una pequeña parte del camino radial que es “ciego” al sensor.

SICK es una compañía alemana basada en la especialización en tecnologías de sensores industriales. SICK específicamente se especializa en soluciones ópticas de automatización, sistemas de seguridad, identificación automática y análisis y procesos de medida. SICK ofrece una variedad de telémetros láser, algunos de ellos más populares como el LMS 200.

El Sick *Laser Measurement Sensor* (LMS) 200 es un sensor de medida de distancia sumamente exacto, que rápidamente se ha unido al la comunidad de robótica, donde es muy común. Su campo horizontal de vista es 180º y puede detectar objetos en distancias hasta 40 o 60 metros, dependiendo de la reflectividad del objeto.

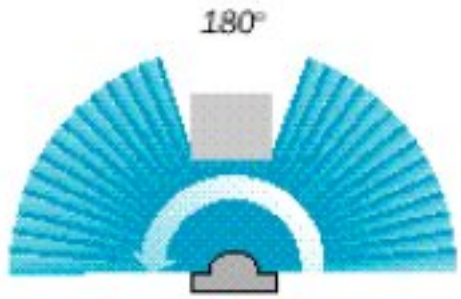

Figura 13: Campo horizontal SickLRF

A pesar de la resolución horizontal de 1º o 0,5º es suficientemente grande, *Sick LRF* tiene una importante desventaja: sólo puede escanear en dimensión horizontal, es decir, en un plano. Un escáner real en 3D será imprescindible en terrenos exteriores, donde el robot se mueve en 3D.

Por esta razón, se ha construido una unidad de inclinación, que es aplicable a la unidad de *Sick LMS* de 60º arriba y abajo cada uno. La parte más importante de la unidad de inclinación es que el motor mueve el escáner, el codificador de la posición absoluta mide el ángulo de la inclinación actual (el cilindro azul) y la electrónica para

el control y la comunicación. Diversas órdenes de movimiento se pueden enviar al controlador (absoluta o inclinación relativa, exploración continua entre dos ángulos y así sucesivamente). El ángulo de inclinación y otros datos se pueden obtener también de esta forma. La posición absoluta del codificador mide el ángulo de inclinación con una resolución de 13 bit. Por tanto, se puede controlar el movimiento del escáner muy de una forma muy exacta y suave.

Este servicio también soporta una vista de los datos LRF a través del protocolo http. En un nodo DSS ejecutando en el puerto 50000, la dirección http://localhost:50000/sicklrf representará el sin procesar la representación xml del estado del servicio, el cual incluye las medidas del LRF.

Visitando http://localhost:50000/sicklrf/cylinder se verá una presentación cilíndrica del los datos del LRF, mientras se visita http://localhost:50000/sicklrf/top representará una vista de arriba hacia debajo de los datos del LRF.

4.3.3. SimpleVision

El servicio *SimpleVision* muestra cómo se puede escribir un servicio que implementa funciones de proceso de imágenes usando una cámara Web. Este servicio funciona con los colores de los objetos, una cara simplificada y detector de gestos de manos. Gráficamente tiene el siguiente aspecto:

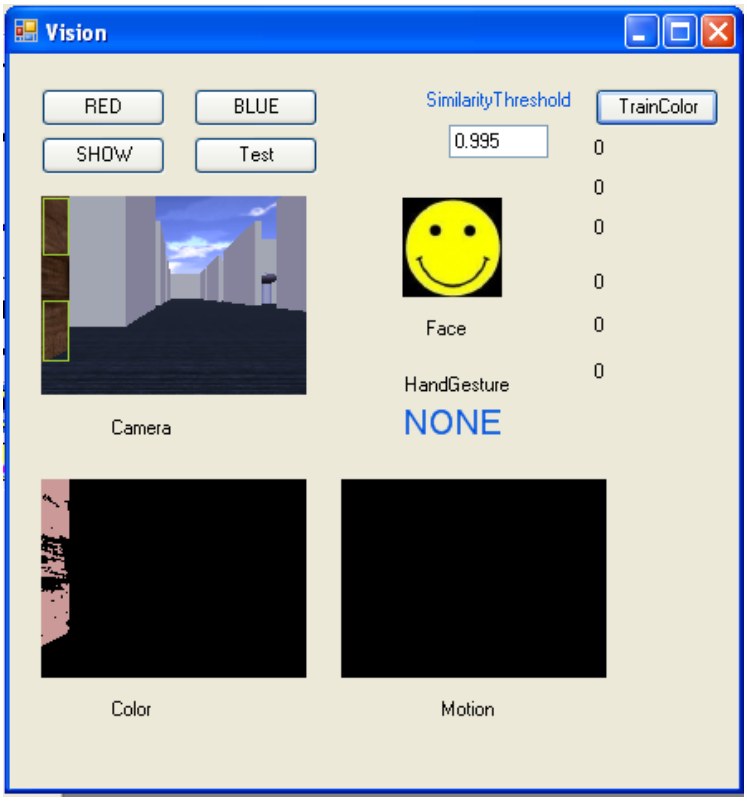

Figura 14. Simple visión

El servicio *SimpleVision* soporta el estado inicial asociado. El estado inicial es usado para configurar el archivo de configuración por defecto para este servicio, que es llamado “simplevision.config.xml”, y éste especifica el intervalo sondeado, los componentes RGB normalizados de un específico blanco de objeto, la similitud de un valor de color y unos valores de área que pueden ser usados en procesamiento de imagen.

Mientras el servicio se está ejecutando, puede detectar el color de un objeto, cara y gestos de manos. El servicio proporciona cuatro tipos de notificaciones. Una para el registro del color, los otros para enviar resultados de detección.

En un servicio, el color de un objeto es considerado como el color de la camiseta del usuario. Esta consideración es también usada para detectar caras y gestos de manos. Para detectar un especificado color de un objeto, el servicio usa

un color normalizado y una medida de similitud entre el actual vector de píxel y el vector y segmentaciones de color registrado.

Para detectar una cara, el servicio usa un filtro de imagen de piel basado en un modelo de color de piel predefinido. Después de alcanzar una imagen de piel, el servicio realiza una segmentación sobre la imagen de piel. Para buscar una región de la cara entre las regiones segmentadas de caras candidatas, el servicio usa una elipse de validación y dos simples constricciones geométricas. La primera constricción es una cara sobre el color de un objeto, y la segunda es que una cara puede ser la parte de la imagen superior de un solo color.

Para detectar gestos de manos, el servicio usa información geométrica sobre una imagen de piel y una imagen de fondo. Esto también usa los resultados de detectar el color de un objeto y una cara. El fondo de la imagen es calculado por substracción de la imagen actual del fondo de la imagen. El fondo de la imagen es captado cuando la película no ha sido detectada entre la actual secuencia de la cámara y la previa por un tiempo especificado en una imagen diferente.

Las operaciones de actualización y notificación proporcionadas son:

- SetObjectTrackingColor.
- NotifyObjectDetection.
- NotifyFaceDetection.
- NotifyHandGestureDetection.

El servicio proporciona un formulario de Windows para registrar un color especificado y mostrar las salidas del proceso de imagen. Se puede registrar un color usando el ratón. Primero, hacer un círculo específico presionando el botón izquierdo del ratón en una posición dada y arrastrar hasta agrandar el tamaño en la imagen de la “cámara” en el formulario. Finalmente, presionar el botón “TrainColor” para registrar un color promedio en una región del círculo. El color registrado es guardad automáticamente en el archivo de configuración **"**simplevision.config.xml**"**

El servicio también proporciona el siguiente método de registro del color de objetos desde otros servicios: “SetObjectTrackingColor**”.**

Se puede enviar el mensaje por correo o usando un método de ayuda después de la asociación con el servicio *SimpleVision*. También permite

suscripciones con otros servicios y enviar tres tipos de notificaciones de resultados de detección a otros servicios.

Además de las notificaciones, el servicio hace llamadas a LogInfo() y LogError() (archivos de información y error respectivamente). Por defecto, estos métodos se envían por la salida del depurador. Esta forma se puede ver usando el panel de control y la apropiada herramienta de desarrollo.

4.3.4. SpeechRecognizer

El servicio *SpeechRecognizer* proporciona diferentes formas de usar un reconocedor de habla, dependiendo de la complejidad del proyecto o conocimiento del usuario.

Este servicio representa el centro del servicio de reconocimiento de habla (en contraposición con el servicio *SpeechRecognizerGui* que ofrece un componente de interfaz de usuario con el servicio central). El servicio central muestra la utilidad de la gramática simple estilo diccionario tan bien como la gramática compleja SRGS (Speech Recognition Grammar Specification), especificada en XML.

El servicio *SpeechRecognizer* por él mismo no expone un interfaz de usuario, lo cual puede hacer duro probar sin escribir el servicio o diagrama VPL. El servicio hermano *SpeechRecognizer* sin embargo muestra para una configuración de una simple gramática o para una subida de complejos SRGS archivos escritos en XML con el significado de una interfaz Web. Se puede empezar una instancia del *SpeechRecognizerGui* una vez que se tenga corriendo el nodo DSS usando un buscador Web y yendo a la página de panel de control.

Una vez que el *SpeechRecognizerGui* está corriendo, se puede buscar en la página Web el servicio. La interfaz Web muestra eventos como el *Speech Detected* (detector de habla) o el *Spech Recognized* (reconocedor de habla) en un área desplazada que puede ser clara.

En la parte de debajo de la página Web de *SpeehRecgnizerGui* se puede definir una gramática. Notar que el *SpeechRecognizer* sólo reconoce palabras y

frases que hay en su gramática. Si la gramática está vacía, entonces no podrá reconocer nada.

4.3.5. TextToSpeech

El servicio simple *TextToSpeech* representa cómo se puede escribir un servicio que interacciona con el API System.Speech .NET.

Muestra una ventana y el estado de habla del objeto (que incluye el último texto hablado) tan bien como otros parámetros de habla. Se pueden cambiar usando la página Web, y también probar la salida del servicio *TextToSpeech* por la introducción de algún texto y presionando el botón “Say” de la ventana.

Las operaciones permitidas en el servicio *TextToSpeech* incluyen lo siguiente:

- **SayText:** reproduce el texto especificado.
- **SayTextynch:** dice el texto especificado pero no envía una respuesta hasta que no se ha terminado de hablar. Si se está usando el servicio con VPL es necesario encontrar esta operación por uno mismo porque la ejecución del programa se puede suspender hasta que se recibe la respuesta por el servicio *TextToSpeech*.
- **SetRate:** establece la velocidad del texto hablado**.**
- **SetVolume:** establece el volumen del sonido.
- **SetVoice:** establece una voz particular.

El servicio *TextToSpech* permite a otros servicios suscribirse a él. Las suscripciones serán notificadas en estados de actualización tan bien como los estados de los visemas y la síntesis de habla. Mientras que los fonemas son la unidad básica acústica del habla, los visemas son la unidad básica visual del habla, es decir, un visema es un patrón de referencia visual de un fonema. Por lo tanto un visema describe una cara en particular y una expresión oral que ocurre junto a la pronunciación de fonemas.

Los mensajes *VisemeNotify* pueden ser usados para ayudar con una visualización una representación facial mientras se habla. El servicio no proporciona una cara simulada, por lo tanto se tiene que escribir el propio código para ella.

Como con muchos servicios, los parámetros del habla pueden ser establecidos en un archivo de configuración. Alternativamente, los parámetros pueden ser modificados a través de mensajes. Los parámetros incluyen:

- **Volume:** Cambios del volumen de la voz. El rango válido es desde cero hasta 100.
- **Rate:** Cambia la velocidad con la que se lee el texto. El rango válido va de -10 a 10.
- **Voice:** Cambia la voz del hablante. El servicio cuenta con varias posibles voces pero este, tan bien como otros parámetros del habla como el tono y el énfasis, puede ser modificado a través del XML en la parte del texto para ser hablado, o en el código variando la frecuencia.

Además de las notificaciones, *TextToSpech* hace llamadas a LogInfo() y LogError(). Por defecto, estos métodos envían la salida por la salida del depurador y se pueden ver con la herramienta de desarrollo apropiada.

5. DESARROLLO DE LA SOLUCIÓN

5. DESARROLLO DE LA SOLUCIÓN

Una vez explicado el funcionamiento de los servicios y el diseño del entorno, se llega al punto más complejo de la aplicación y es adaptar el movimiento del robot al entorno y que se mueva únicamente por una ruta prefijada.

El problema que se plantea en este punto es cómo conseguir que el robot se mueva por un camino prefijado y que pueda ir de una habitación a cualquier punto del hotel de un modo directo. La manera más optima de resolver esto es aplicando un algoritmo de caminos mínimos, como es el algoritmo de Dijkstra. Es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo dirigido y con pesos en cada arista.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen al resto de vértices que componen el grafo, el algoritmo se detiene.

El algoritmo es una especialización de la búsqueda de coste uniforme, y como al, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el coste general del camino al pasar por una arista con coste negativo).

Para ello, se tendrán que crear las clases necesarias para implementar la estructura de grafos y poder aplicar el algoritmo de búsqueda de camino mínimo.

5.1. Modificación de servicios

En base a la reutilización de servicios, se ha usado el *Follower* proporcionado por MSRS para adaptarlo según las necesidades requeridas. Primeramente se ha prescindido de los servicios *Contact Sensor*, *SickLRF* y *SimpleVision* por los siguientes motivos:

- Contact Sensor: este servicio asociado al *Follower* permite evitar obstáculos en el entorno. En este caso se supone que el entorno no contiene ningún obstáculo, por tanto no hará falta evitarlo.
- SickLRF: La exploración del entorno no es necesaria, puesto que el robot tendrá un camino virtual por el que se moverá y del que no saldrá en ningún momento de la ejecución.
- SimpleVision: este servicio se utiliza para interactuar con el robot en un estado real, reconoce gestos, caras y colores de ropa. Puesto que en todo momento se trabaja con el entorno simulado, este servicio no es necesario.

5.1.1. Servicio Guía

A parte de la creación de varias clases para implementar las funcionalidades deseadas, se ha modificado la interfaz gráfica del *Follower*, pues tiene que contemplar la opción de moverse por todas las habitaciones creadas, por tanto la ventana inicial no sirve. El estado final del formulario para manejar el servicio se muestra en la figura 15.

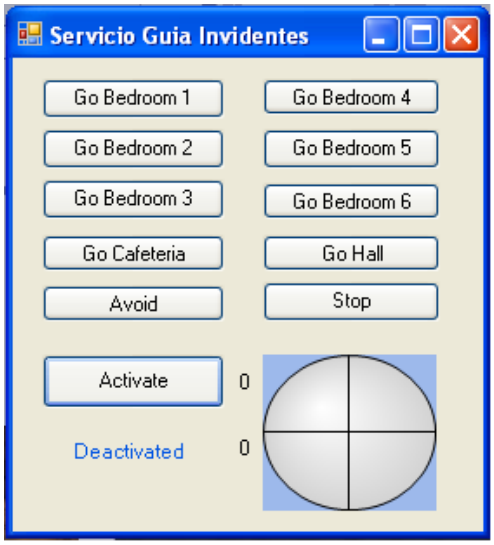

Figura 15. Apariencia final del Follower.

También se han añadido nuevos estados lógicos de conducta del robot, tantos como habitaciones hay en el entorno (GoBedroomOne, GoBedroomTwo…etc).

5.1.2. Entorno final

Para conseguir el entorno final ha sido necesaria la creación de multitud de entidades, ya que el entorno de partida era muy simple.

Todo el plano del entorno virtual está delimitado por las coordenadas X,Y,Z, por tanto, una vez que se tienen todas las entidades diseñadas, únicamente hay que colocarlas en la posición deseada y guardar la escena, que no es otra cosa el archivo .xml con todos los detalles del entorno y todas las entidades que contiene. Un ejemplo de este archivo se muestra en la figura siguiente:

```
- <SerializedEntities>
  - <SingleShapeEntity xmlns:sim="http://schemas.microsoft.com/robotics/2006/04/simulation.html"
      xmlns:physm="http://schemas.microsoft.com/robotics/2006/07/physicalmodel.html"
      xmlns:phys="http://schemas.microsoft.com/robotics/2006/04/simulation/physics.html"
      xmlns="http://schemas.microsoft.com/robotics/2006/04/simulationengine.html">
    - <sim:State>
        <sim:Name>lampara_i_5</sim:Name>
      - <sim:Assets>
          <sim:Mesh>Spotlight.obj</sim:Mesh>
        </sim:Assets>
      - <sim:Pose>
        - <physm:Position>
            <physm:X>11.2</physm:X>
            <physm:Y>1.8</physm:Y>
            <physm:Z>19.5</physm:Z>
          </physm:Position>
        - <physm:Orientation>
            <physm:X>0</physm:X>
            <physm:Y>0</physm:Y>
            <physm:Z>0</physm:Z>
            <physm:W>1</physm:W>
          </physm:Orientation>
        </sim:Pose>
```

Figura 16. Ejemplo de archivo .xml

A continuación se muestran unas imágenes del hotel resultante, que cuenta con seis habitaciones con baño, cafetería y hall.

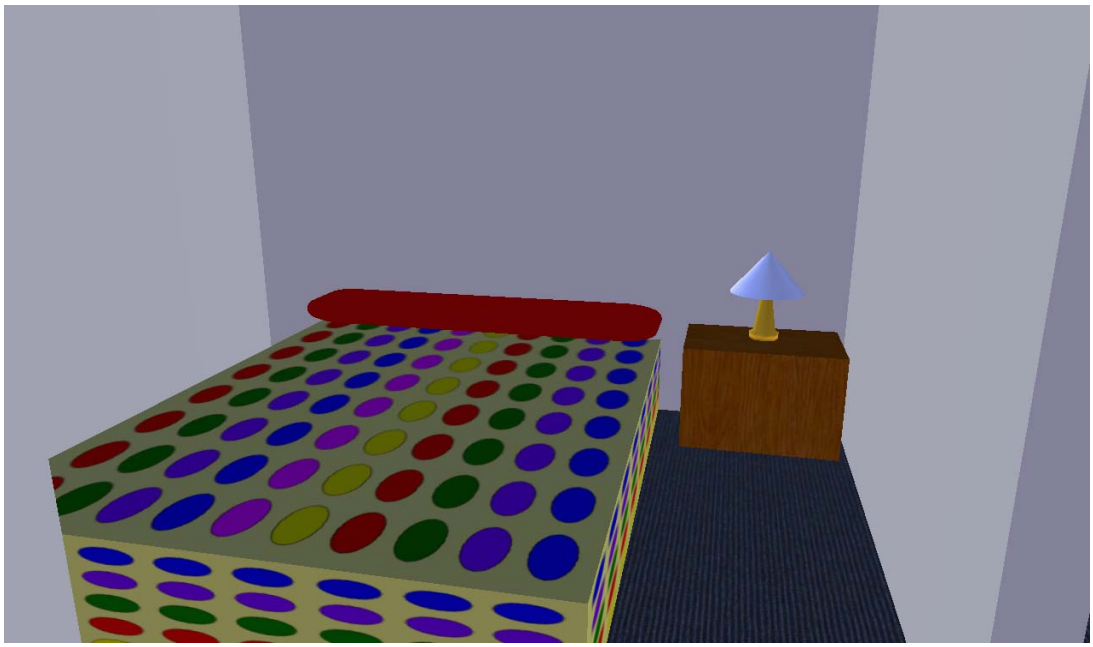

Figura 17. Habitación simple

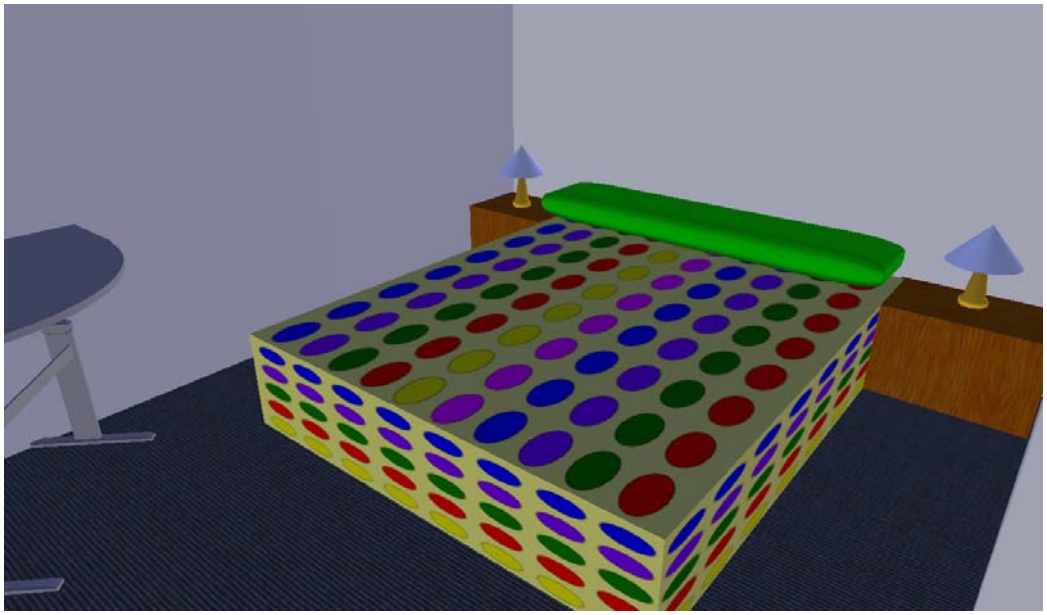

Figura 18. Habitación de matrimonio

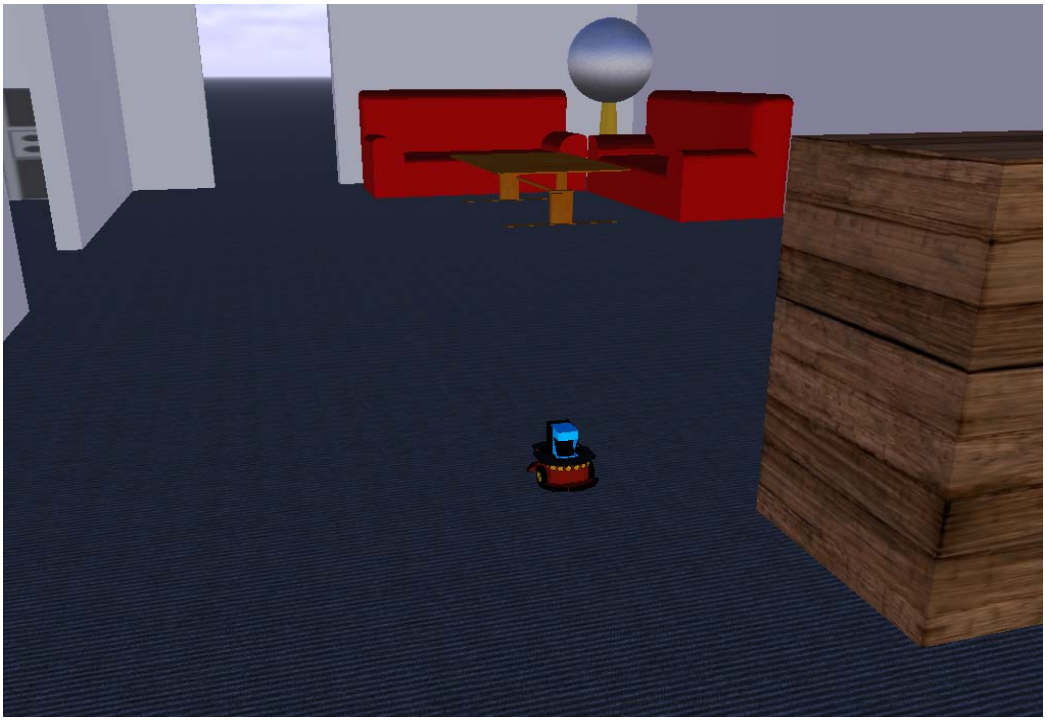

Figura 19. Hall

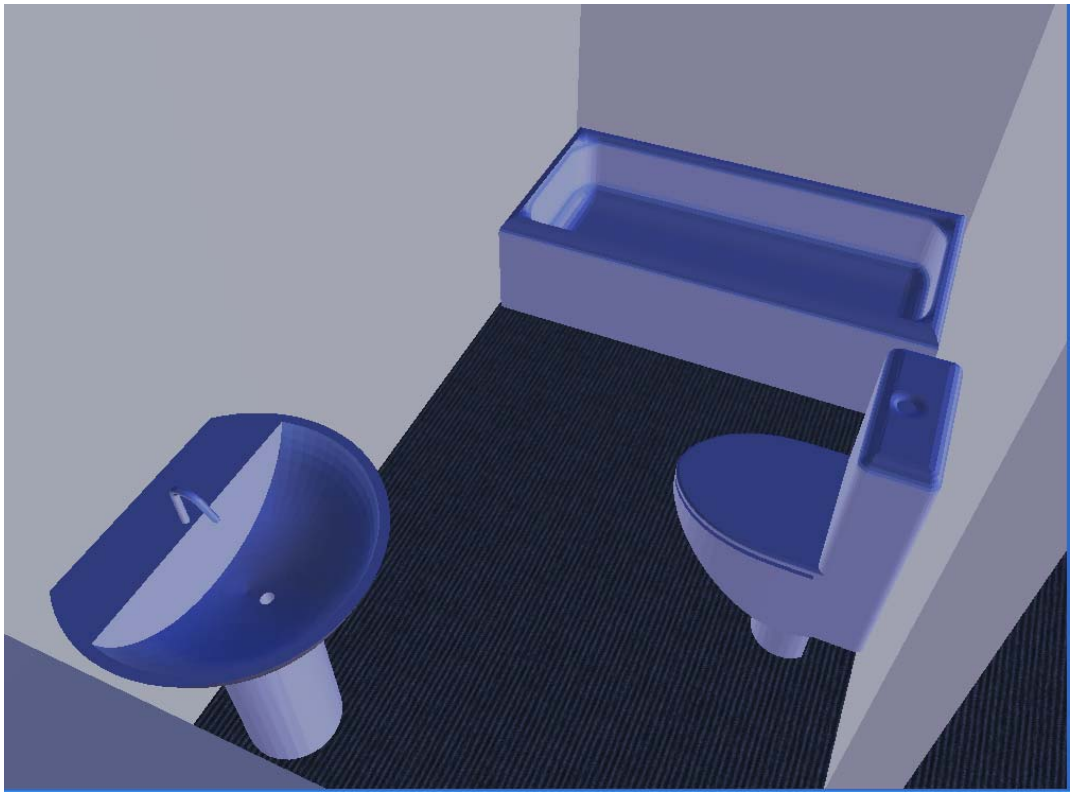

Figura 20: Baño

Figura 21. Distribución habitaciones.

Figura 22. Cafetería

5.1.3. Cambios en TextToSpeech

En este servicio lo único que se ha modificado son las nuevas “órdenes” que el robot debe entender. Por tanto se han añadido las siguientes frases:

- GoBedroomOne.
- GoBedroomTwo.
- GoBedroomThree.
- GoBedroomFour.
- GoBedroomFive.

- GoBedroomSix.
- GoHall.
- GoCafeteria.

5.2. Descripción de clases para el movimiento

Para contemplar el movimiento del robot se parte de la estructura de grafos, explicada en el punto 2.5.1, pero para ello, es necesaria la creación de diferentes clases que implementen dicha estructura.

5.2.1. La clase Node

La clase *Node* contiene un objeto que puede ser usado para almacenar cualquier información asociada a un nodo. Además es necesaria una forma de identificar fácilmente a los nodos, por tanto la clase también contiene la propiedad *Key*, que servirá para identificar únicamente a cada nodo.

Ya que se usa la técnica de lista adyacente para representar el grafo, cada instancia de *Node* necesita tener una lista de sus nodos vecinos. Si el grafo usa aristas dirigidas, la lista adyacente necesitará también almacenar el peso de cada arista. Para manejar esta lista adyacente, es necesario crear una clase *AdjacencyList* para crear una propiedad de este tipo en la clase *Node.*

5.2.2. Las clases AdjacencyList y EdgeToNeighbor

Un nodo contiene una clase *AdyacencyList* (lista adyacente), que es una colección de las aristas hacia los nodos vecinos. Desde una *AdjacencyList* se

almacena una colección de aristas, pero primero es necesario crear una clase que represente a la arista. Esta clase se llamará *EdgeToNeighbor*, que modela una arista y extiende a los nodos vecinos. Ya que se puede necesitar asociar un peso a una arista, la clase *EdgeToNeighbor* necesita dos propiedades: cost (un entero que indica el peso de la arista) y *Neighbor* (un objeto *Node* referencia).

La propiedad de *Neighbor* de la clase *Node* expone la variable interna *AdjacencyList* del nodo. Notar que el método *Add()* de la clase *AdjacencyList* es interno de la clase, sólo las clases dentro de los archivos del mismo ensamblado pueden añadir una arista. Esto hace que sólo el desarrollador que use la clase *Graph* pueda modificar la estructura del grafo por los miembros de dicha clase y no por la propiedad *Neighbor*.

A parte de las propiedades, *Data* y *Neighbor*, la clase *Node* necesita proporcionar un método que permita al desarrollador manipular la clase grafo para añadir aristas desde el mismo nodo a otro vecino. Recordar que con la lista de adyacencia se cataloga, si existe una arista no dirigida entre los nodos u y u, entonces u tendrá una referencia a v en su lista de adyacencia y v tendrá una referencia de u en su lista. Los nodos sólo deberían ser responsables de mantener su lista de adyacencia, y no de los otros nodos del grafo.

Para hacer más fácil el trabajo de añadir una arista entre dos nodos de la clase *Graph*, la clase *Node* contiene un método para añadir aristas dirigidas desde el mismo hacia otro vecino. Éste método, *AddDirected*(), toma la instancia de un nodo y un peso opcional y crea una instancia de *EdgeToNeighbor* y la añade a la lista de adyacencia del nodo.

5.2.3. La clase Graph

Ésta clase contiene un array de instancias de nodos (*NodeList*), que a su vez cada uno contiene la información sobre su posición (objeto *data*), nombre (*key*) y una lista adyacente (*AdjacencyList*) donde hace referencia a sus nodos vecinos.

Notar que con la técnica de la lista de adyacencia, el grafo mantiene la lista de los nodos. Cada nodo a su vez, mantiene una lista de adyacencia de nodos.

Entonces, en la creación de la clase *Graph* es necesario tener una lista con todos los nodos. Se puede optar por usar un *ArrayList* para mantener esta lista, pero es más eficiente usar un *Hashtable*. Aquí, el es una aproximación más sensible porque en los métodos que se usan para añadir una arista al grafo, es necesario asegurarse de que los dos nodos especificados para añadir la arista deben existir en el grafo. Con un *ArrayList* se tendría que hacer una búsqueda lineal por todo el array hasta encontrar ambos nodos, con un *Hashtable* se hace una consulta con tiempo constante.

La clase *NodeList*, contiene los métodos de *Add()* y *Remove()* para añadir o eliminar instancias del nodo al grafo. También contiene el método *ContainsKey*() que determina si para una clave en particular, existe el nodo que la contiene en el grafo.

La clase *Graph* contiene la propiedad pública de *Nodes*, que es del tipo *NodeList*. Además, tiene diversos métodos para añadir aristas dirigidas o no dirigidas, con peso o sin peso, entre dos nodos existentes en el grafo. Por ejemplo, el método *AddDirectedEdge*() crea una arista desde el primer nodo al segundo y un peso opcional. Además, en todos estos métodos para añadir aristas, la clase *Graph* tiene el método *Contains*() que devuelve un booleano indicando si un nodo en particular existe en el grafo o no.

5.2.4. Representación del camino usando una lista adyacente

Para representar el camino por donde se deberá mover el robot por el hotel se ha utilizado la clase *Graph*. La representación del grafo del hotel se puede ver en la figura 22, donde cada nodo está situado en un lateral de cada una de las puertas de las diferentes habitaciones, cafetería y recepción.

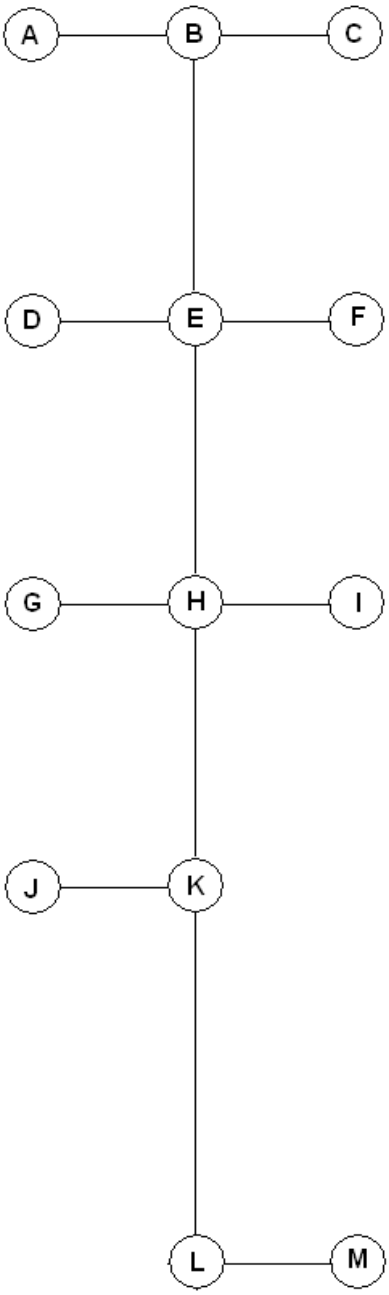

Figura 23. Representación del grafo del hotel.

Como se ha comentado anteriormente, cada nodo tiene una lista adyacente donde se hace referencia a sus nodos vecinos. En la figura 24**,** se muestra una representación de cada nodo con su lista adyacente:

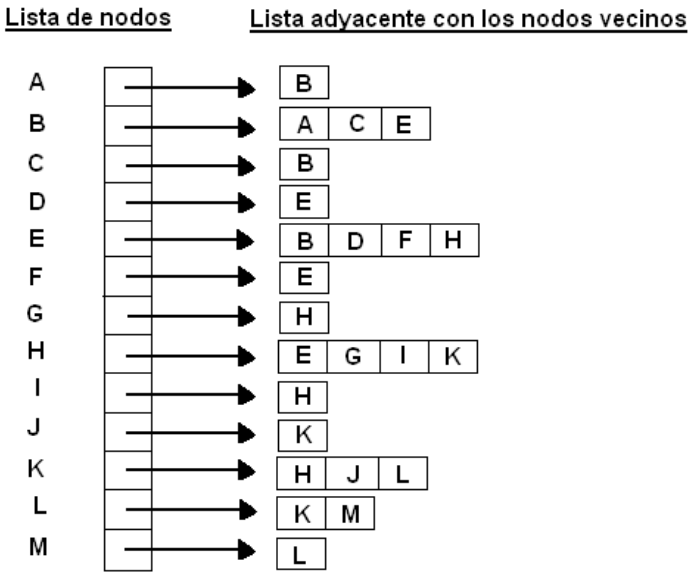

Figura 24. Listas adyacentes de los nodos vecinos de cada nodo.

Notar que con un grafo no dirigido, la representación de una lista de adyacencia duplica la información de la arista. Por ejemplo el nodo A tiene a B en su lista de adyacencia y el nodo B tiene a A en su lista de adyacencia también.

Cada nodo tiene precisamente tantos nodos en su lista de adyacencia como vecinos, por tanto esta estructura de datos es una forma eficiente en términos de espacio de representar el grafo. Nunca se almacenan más datos de los que se necesitan, en concreto, para un grafo con V nodos y E aritas, usando una representación basada en listas adyacentes, se requieren V+E instancias de *Node* para un grafo dirigido y V+2E instancias de *Node* para un grafo no dirigido.

Aunque en la figura 24 no aparece, una lista adyacente se puede usar para representar grafos con peso. La única diferencia es que para cada lista adyacente, cada instancia de *Node* necesita almacenar el coste de la arista.

5.3. Consecución del movimiento del robot. Algoritmo de Dijkstra

En este punto se detallará el engranaje de todas las piezas anteriores para conseguir que el robot se mueva por el grafo establecido y se explicará en qué consiste recorrerlo utilizando un algoritmo de camino mínimo.

Para la creación del grafo, se utiliza un fichero donde se encuentran todos los datos referentes a cada nodo, como son el/los nodo/s vecino/s, la distancia entre los nodos, y las coordenadas en el entorno de simulación (x,y,z).

Se crea un elemento *Graph* y dos elementos *Hashtable*, uno para almacenar la menor distancia, y otro para almacenar el camino.

Una vez creada una instancia de la clase *Graph*, se llena el grafo con los datos del archivo. Lo siguiente es añadir los nodos en el grafo. Esto implica invocar al método *AddNode()* para cada nodo que se quiera añadir, con la clave (nombre) y el objeto *data* como un nuevo objeto de tipo Coordenada (x,y,z), ya que se conocen todos los datos una vez cargado el fichero.

Una vez construido el grafo, se plantean algunas cuestiones, como ¿cómo se va de un nodo a otro con el menor número de saltos? La respuesta a esta pregunta se obtiene analizando el comportamiento de algoritmos de caminos mínimos. En este caso, el camino es único, ya que no existen ciclos en el grafo.

Edgar Dijkstra, uno de los informáticos más conocidos de todos los tiempos, inventó el algoritmo más comúnmente usado para encontrar el menor camino de un nodo origen al resto en un grafo dirigido con peso. Este algoritmo trabaja manteniendo dos tablas, una tabla de distancia, que guardará la mejor distancia actualizada del nodo fuente al resto, y una tabla de ruta, que para cada nodo n indica qué nodo fue utilizado para alcanzar n para conseguir la mejor distancia.

Inicialmente, la tabla de distancias tiene para cada registro fijado un valor elevado (infinito positivo), salvo para el nodo de inicio, que tiene una distancia a él mismo de cero. Las filas del vector de la ruta se establecen a *null*. Además, hay una

colección de nodos Q que se examina para cada iteración hasta que no tenga elementos en ella. Inicialmente esta colección tendrá todos los nodos del grafo.

El algoritmo procede seleccionando y eliminando el nodo de Q que tenga el menor valor en la tabla de distancias. Si a este nodo seleccionado se le llama n y su valor en la tabla de distancias es d, para cada una de las aristas de n, se comprueba si d más el coste para obtener n de un vecino es menor que el valor para ese vecino en el vector de distancias. Si lo es, es que se ha encontrado un mejor camino de alcanzar a ese vecino, y las tablas de ruta y distancia se actualizarán según esta elección. Se itera hasta que no queden nodos en Q.

La aplicación de este algoritmo en el proyecto es la siguiente: al pulsar cualquier botón del Form “Servicio Guia Invidentes” se obtiene la posición del robot actual (y en consecuencia el nodo), y se ejecuta el algoritmo con el nodo origen y el destino. El resultado se almacena en dos objetos *Hashtable*, uno para la distancia que conlleva el movimiento del punto de partida hasta el final, y el otro con la ruta que debe seguir el robot.

6. DISEÑO DE EXPERIMENTOS Y PRUEBAS

6. DISEÑO DE EXPERIMENTOS Y PRUEBAS

6.1. Diseño de experimentos

En la resolución de cualquier problema de ingeniería, se deben solventar una serie de errores hasta la obtención de los resultados óptimos esperados. Estos errores se encuentran una vez que se pasa una batería de pruebas, cuyo alcance se detalla a continuación.

- **FASE I: Entorno.**

  En esta primera fase de pruebas se tiene que comprobar la integridad del entorno creado modificando las constantes de los materiales para que su unión no sea incompatible. En el modo edición se puede unir todo sin problemas, pero en tiempo de ejecución pueden moverse o "volar" objetos del entorno debido a ciertas propiedades.

- **FASE II: Movimiento.**

  Al modificar el entorno original, a priori no se sabe el comportamiento y respuesta que tendrá el robot respecto a estos cambios. Se deberá probar diversos tipos de movimiento y analizar su respuesta para entender el funcionamiento y aplicarlo a la consecución del objetivo, que es guiar a una persona por el hotel por una ruta prediseñada.

6.2. Pruebas

En este punto se detallarán los errores encontrados al depurar la aplicación con la batería de pruebas citadas en el apartado anterior.

6.2.1. Fase I: Entorno

Como se ha comentado en puntos anteriores MRDS proporciona el servicio del entorno virtual del que se parte para la construcción del hotel. El entorno que proporciona el servicio *Follower* es un entorno sencillo, cuatro paredes y obstáculos de tipo caja, bolas y conos. Se puede ver mejor con la siguiente imagen:

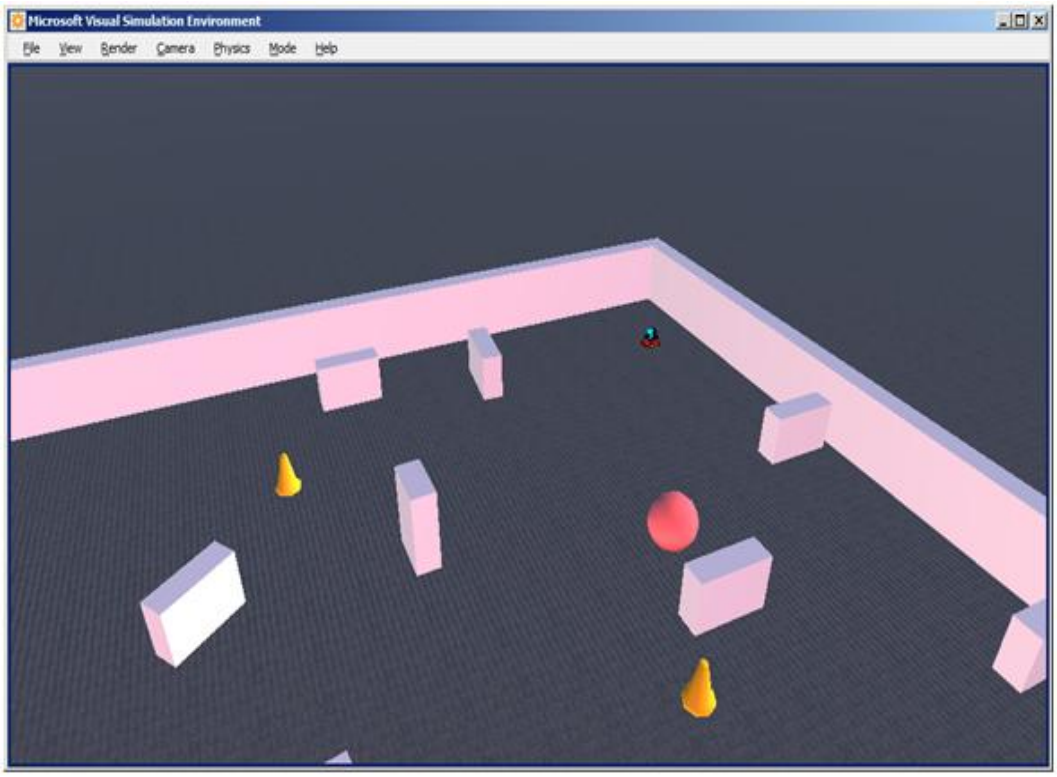

Figura 25. Entorno base

El entorno actual tiene el siguiente aspecto:

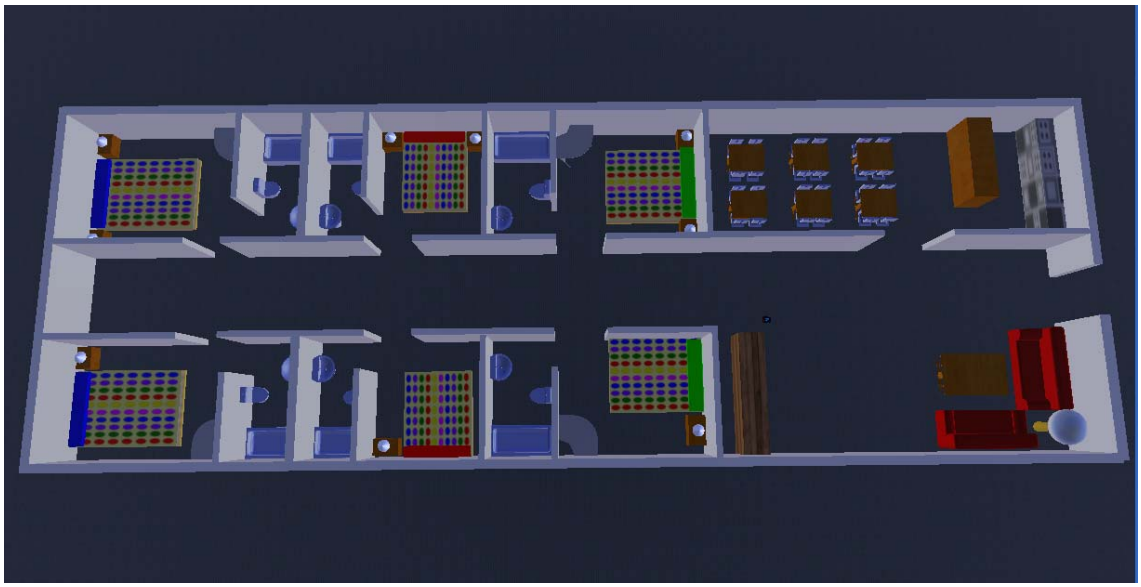

Figura 26. Entorno final

La modificación hasta obtener el hotel tal y como se muestra en la imagen 26, ha sido enorme, tanto de ajustar todas las paredes para que no se caigan como la creación de nuevas entidades diseñadas con SolidWorks adaptadas para MRDS. En el modo edición, es únicamente donde se puede modificar el entorno de la simulación. Una vez construido el entorno final, se genera el .xml necesario para cargar la simulación.

Los principales problemas de la construcción del entorno son los siguientes:

- **Constantes de los materiales**

  Para recrear el hotel, han sido necesarias muchas paredes, tal y como se ha especificado en el punto 4.2. (Entorno de Simulación). El problema que se ha encontrado en el entorno, es que al guardar un escenario desde el modo edición para posteriormente abrirlo en una nueva ejecución, las paredes volaban porque las constantes de los materiales eran incompatibles. Por tanto había que colocarlas otra vez y probar modificando los parámetros de los materiales hasta encontrar los adecuados.

- **Movimiento de paredes**

Una vez solucionado el problema del punto anterior, se encontró otra adversidad en las paredes del mobiliario. Una vez guardado un escenario, a la hora de abrirlo, algunas de las paredes habían modificado su posición en ángulos respecto al eje vertical y horizontal. Por tanto, todas las habitaciones aparecían torcidas e inclinadas las paredes respecto a la vertical. Tras probar numerosas opciones, se consiguió mantener la orientación de las paredes guardando los cambios producidos en los materiales después de la colocación correcta.

- **Visualización de las entidades creadas con SolidWorks**

Tal y como se ha comentado en el punto 4.2.3. (en el apartado referente a SolidWorks) las entidades creadas en SolidWorks, deben convertirse al formato .obj para poder introducirlas en el entorno simulado a través de la aplicación Blender.

Hay dos tipos de archivos .obj para convertir desde Blender:

Con el primero, las entidades creadas no tenían forma ni color, aparecían como cubos blancos.

Con el segundo, las entidades aparecían con la silueta, pero sin forma, ni texturas. Probando diversas opciones de importación, se dio con la importación adecuada, que es haciéndolo en modo escena. De esta forma, los archivos tiene extensión _Scene.obj, válidas para MSRS y visualizadas a la perfección, tal y como se han diseñado en SolidWorks.

Una vez conseguido el entorno final, se pasa a la fase II de pruebas que es la referente al movimiento, explicada con detalle en el siguiente punto.

El primer problema encontrado es que con el entorno final el robot no se mueve, ni utilizando los métodos existentes en el *Follower* ni aplicando potencia directamente al servicio Drive, encargado del movimiento de las ruedas.

Éste entorno final ha sido creado en varias fases, primero se creó la distribución de paredes. En una etapa se crearon los muebles a partir de las entidades proporcionadas por el entorno de simulación y finalmente, se introdujeron las entidades creadas con SolidWorks. Tras varias pruebas, se observa que en el único entorno que se mueve el robot es en el original, entonces se llega a la conclusión de que si se modifica el entorno en una única fase, el robot debe funcionar.

Efectivamente, al abrir el entorno original, modificarlo hasta conseguir el entorno final (proceso realizado más rápidamente que la primera vez por el grado de aprendizaje obtenido en la primera intervención) y guardándolo, se consigue que el robot se mueva con el DashBoard implementado en el formulario de navegación del servicio Guía Invidentes.

6.2.2. Fase II: Movimiento

El principal problema de este proyecto ha sido el movimiento del robot. En el servicio *Follower*, se usa el servicio asociado *Drive* para controlar los movimientos del robot, como el estado y la velocidad, pero no contempla la opción de obtener la posición del robot en el entorno, ya que a priori el robot se movía libremente por el entorno según las indicaciones recibidas.

Por tanto ha sido necesaria la modificación del estado del *Follower* para contemplar aspectos de posicionamiento en el entorno, entre otras cosas, añadiendo diversas referencias (ver código). Añadiendo al proyecto las referencias *Microsoft.Robotics.Simulation.Engine* y *Microsoft.Robotics.Simulation.Physics*, se consigue obtener la posición del robot en cada momento.

Una vez que se tiene constancia de la posición del robot en el entorno, utilizando los métodos del *Follower* original para mover, girar y parar el robot no se consiguen resultados, porque aunque el robot esté situado en una cierta posición, para el servicio *Drive* siempre parte en la posición 0,0,0 y no había coherencia entre posición real y la posición de las ruedas del *Drive*.

La solución adoptada para este problema ha sido la implementación de nuevos métodos para coordinar el movimiento del robot.

Aplicando directamente potencia al motor e indicando la distancia que se tiene que mover se comprueba que los resultados son los esperados. El problema surge cuando se envía la orden repetidas veces al robot de moverse y de girar (para la descripción de movimiento sobre el grafo de la figura 23), ya que se produce un solape de movimientos que anulan al robot. Se piensa entonces en poner un tiempo de espera entre un movimiento y un giro, porque si recordamos el esquema del grafo del hotel de la figura 23, se puede unir en un solo paso un desplazamiento y giro para alcanzar un nodo. Por ejemplo, si se quiere ir del nodo A al nodo G, la ruta sería la siguiente: de A a B, giro 90º, de B a E, no giro (porque es línea recta), de E a H y giro 90º, de H a G y giro 180º para posicionar el robot en la espera de la orden del siguiente movimiento.

Si se combina movimiento y giro en un método de tipo iterador, se puede controlar el tiempo de espera de un movimiento a otro tantas veces como nodos existan en la ruta calculada con los nodos de origen y destino. De esta forma el robot llega a su destino de una forma correcta.

El último problema encontrado respecto al movimiento de robot es la cancelación sin motivo aparente del movimiento. Esto es debido a que el entorno de simulación está diseñado con una alta fidelidad y con un alto grado de similitud con la realidad. Cualquier parada brusca, inclinación del terreno o cosas que no parecen que un entorno de simulación pueda captar, trascienden en una parada del movimiento del robot como puede darse en un caso real.

7. CONCLUSIONES

7. CONCLUSIONES

El hecho de solucionar un problema de ingeniería, siempre es un reto en términos de optimización y eficiencia entre otros, pero cuando se carecen de conocimientos previos en la materia resulta mucho más complejo, ya que es necesaria una dedicación mucho más exhaustiva para la comprensión de toda la estructura de las herramientas que se van a utilizar. Y éste, es sin duda, el mayor logro de este trabajo.

La finalidad de este proyecto es “engranar” diferentes herramientas y servicios para proporcionar un entorno simulado donde un robot Pioneer3DX sirva como guía a una persona invidente.

El motor de trabajo es Microsoft Robotics Developers Studio. Es una herramienta muy potente que ha permitido la creación de esta aplicación y módulos en tiempo de ejecución orientado a servicios a través de una aplicación Windows, en este caso la ventana para manejar al robot por los diferentes puntos del hotel. Se ha conseguido que múltiples tareas interactivas simultáneas se realicen en tiempos distintos mediante programación asíncrona concurrente. Por ejemplo, el robot está constantemente actualizando su estado mientras que a su vez puede moverse, utilizar la cámara y obtener la posición dentro del entorno.

Para que todo esto funcione, es necesario implementar los servicios y módulos con una herramienta de desarrollo. Antes de elegir la herramienta para desarrollar la aplicación, se han analizado tanto Visual Programming Languaje (VPL), proporcionada por MRDS, como Visual C#.

Para comentar los pros y contras de cada una de ellas, se resumen en la siguiente tabla:

	PROS	CONTRAS
VPL	-No se necesitan conocimientos avanzados de programación. -Programación visual que permite una comprensión fácil de la coordinación entre servicios	-Poco potente para aplicaciones avanzadas. -No es intuitiva para encontrar errores en tiempo de ejecución.
C#	-Amplio lenguaje para desarrollar aplicaciones .NET. -Depurador sencillo y eficaz. -Biblioteca de clases .NET. -Seguro en el tratamiento de tipos	-Necesarios conocimientos avanzados de programación -Más complejo

Después del análisis de estas dos herramientas, se concluye que Visual C# proporciona mucha más potencia de diseño, dada la amplitud de funcionalidades que se pueden implementar.

Notar que, uno de los motivos para la elección de C# como herramienta de desarrollo es que se parte de conocimientos previos en programación orientada a objetos (Java y Visual Basic.NET), y aunque ha sido necesaria el estudio de programación en Visual C# para obtener los resultados deseados, es preferible utilizar esta herramienta por su amplio lenguaje para desarrollar aplicaciones .NET.

A parte del diseño de la aplicación con Visual C# y la comprensión y utilización de MRDS, ha sido necesario el aprendizaje de la herramienta SolidWorks para modelar los diferentes elementos del entorno de simulación.

Se han tomado dimensiones reales para todas las piezas y un material homogéneo e isótropo, ya que las piezas únicamente sirven para decoración y no se va a realizar ningún análisis en ellas, como pueden ser resistencia, elasticidad, etc.

A partir de un croquis en un plano en dos dimensiones se utiliza la herramienta de extrusión para obtener la figura en tres dimensiones y mediante cortes, redondeados, pulido de aristas y demás herramientas de SolidWorks se han modelado cada una de las piezas que conforman el entorno.

Como resultado del engranaje de los diferentes servicios, se ha conseguido que el robot se mueva por el entorno. Para ello ha sido necesaria la implementación de diferentes clases y el estudio de estructuras de datos proporcionadas por la biblioteca de clases de Visual C# para aplicar un algoritmo de caminos mínimos. Dado que se han considerado las diferentes habitaciones del hotel como un grafo, la solución óptima es implementar un algoritmo de caminos mínimos para recorrerlo. En tiempo de ejecución se obtiene la posición del robot en el entorno constantemente y se controla que el movimiento se realiza por los caminos establecidos del grafo y no fuera de ellos.

El aspecto más importante en el desarrollo de la aplicación para controlar el robot es la coordinación entre múltiples tareas interactivas y simultáneas, conocido como programación asíncrona y concurrente.

La correcta secuencia de interacciones o comunicaciones entre procesos y el acceso coordinado de recursos que se comparten por todos los procesos o tareas son las claves para realizar el control del robot. Al mismo tiempo que el robot se mueve por el entorno simulado, es capaz de realizar otras actividades como leer datos del láser, de los *bumpers* y hablar mediante el servicio *TextToSpeech*.

Una última conclusión que se obtiene de este trabajo es la fidelidad existente entre el entorno virtual y la realidad. El comportamiento del robot en el entorno es como si estuviera en uno real, por ejemplo, si impacta con algún objeto retrocede hasta el punto de volcar, si tiene un obstáculo que le impide el movimiento patinan las ruedas, etc. Con ese comportamiento descrito se descubre la potente

herramienta proporcionada por MRDS que es el entorno virtual, ya que representa con una alta fidelidad el mundo real y puede ser muy útil en pruebas de diversos ámbitos antes de reproducirlas con un robot real.

8. PROYECTOS FUTUROS

8. PROYECTOS FUTUROS

Este proyecto es sólo una pequeña pieza del complejo puzzle que se podría hacer en este campo. Una vez solucionado el problema del movimiento por un entorno delimitado mediante un camino virtual delimitado por coordenadas, los siguientes serían poder moverse con total autonomía por el entorno conocido, poder avisar y salvar obstáculos (navegación local) y ya a más alto nivel, solucionar el problema del SLAM: construcción de un mapa de un entorno desconocido y localizarse en él. (Todo esto explicado más detalladamente en el punto 2.3.5).

Una de las posibles mejoras futuras es partir de un entorno conocido (el hotel) con posibilidad de encontrar obstáculos temporales y solventarlos para volver a la ruta original, combinando con el servicio *TextToSpeech* para avisar a la persona invidente que hay un obstáculo y las medidas que se tienen que tomar para solventarlo. En definitiva, se trataría de dotar al robot de un sistema de navegación local combinado con el de navegación global implementado en el presente proyecto.

Otra opción más compleja es utilizar una navegación basada en mapas construidos por el usuario, ya que el mapa del hotel es un entorno delimitado. De esta forma no sería necesaria la utilización del algoritmo para encontrar el camino mínimo, puesto que el robot se movería libremente por el entorno.

Primero se debería recorrer el hotel en modo aprendizaje, para detectar y ubicar las diferentes habitaciones en el mapa que el robot va construyendo, y posteriormente, la localización en cualquier punto del hotel solicitada por la persona invidente. Todo esto combinado con el servicio *TextToSpeech* para indicar al usuario con el máximo realismo posible lo que está pasando a su alrededor.

A estas mejoras propuestas se les puede añadir el reconocimiento facial y de gestos del robot hacia la persona y el servicio *SpeechRecognize*r para poder indicarle al robot la habitación donde se quiere ir por medio del habla.

Éstas son las mejoras más próximas y factibles de este proyecto, pero existen diversidad de funciones que pueden ser implementadas mediante la aplicación de diferentes técnicas del campo de la robótica.

REFERENCIAS

REFERENCIAS

- **[1]**. Ministerio de Educación y ciencia. Secretaría general de política científica y Tecnológica. Nota de prensa. www.mepsyd/es. 2007
- **[2].** Libro Blanco de la Robótica. CEA (Comité Español de Automática) ,2008.
- **[3].** Robots Autónomos y aprendizaje por refuerzo, Farid Fleifel Tapia, 2002.
- **[4].** Robot, Wikipedia. Noviembre 2008
- **[5].** Ver [2]
- **[6].** Ver [2]
- **[7].** Brooks R., "Cambrian Intelligence. The Early History of the New AI", MIT Press, 1999.
- **[8].** Martínez J.L. (1994) "Seguimiento Automático de Caminos en Robots Móviles".Tesis Doctoral. Universidad de Málaga.
- **[9].** Shin D. H., Singh S. (.990) "Path Generation for Robot Vehicles Using Composite Clothoid Segments". The Robotics Institute, Carnegie-Mellon University. Internal Report CMU-RI-TR-90-31.
- **[10].** Cox J.I. (1991)"Blanche-An Experiment in Guidance of Autonomous Robot Vehicle". IEEE Transactions on Robotic and Automation. Vol 7, No 2, abril de 1991, pp 193-204.
- **[11].** Nelson L. W., Cox I. J. (1990) "Local Path Control for an Autonomous Vehicle". Autonomous Robot Vehicles. Editores I.J. Cox y G.T. Wilfong. Springer-Verlag. pp 38-44.
- **[12].** Levi P. (1987) "Principles of Planing and Control Concepts for Autonomous Mobile Robots". Proc. of 1987 IEEE International Conference on Robotics and Automation. pp 874-881.
- **[13].** G. DeSouza and A. C. Kak, "Vision for mobile robot navigation: a survey," IEEE Transactions on Pattern Analysis and Machine Intelligence February, vol. 24, no. 2, pp. 237–267, 2002. [Online]. Available: http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/trans/tp/.

- **[14].** J. Borenstein, H. Everett, and L. Feng, Navigating Mobile Robots: Systems and Techniques, Wellesley, Ed. AK Peters, 1996. [Online]. Available: http://www-personal.umich.edu/_johannb/shared/pos96rep.pdf.
- **[15].** www.**solidworks**.com
- **[16]**. Simulation Import Tutorials. Information on how to import your 3D robot model into MSRS http://channel9.msdn.com/wiki/simulationimporttutorials.
- **[17]**. Blender, Wikipedia. http://es.wikipedia.org/wiki/Blender
- **[18]**. Ver [16]

BIBLIOGRAFÍA

BIBLIOGRAFÍA

- Robots Autónomos y aprendizaje por refuerzo, Farid Fleifel Tapia, 2002
- Manual de desarrollo de un sistema de control robótico con un lenguaje de programación visual.Trabajo dirigido, Paula Berrio Martínez, 2008. Departamento de informática. Universidad Carlos III de Madrid.
- Libro Blanco de la Robótica. CEA (Comité Español de Automática) ,2008.
- Entorno de Simulación Visual, Ana Santos, Raúl Arrabales, 2008.
- http://www.conscious-robots.com/en/forums-./foro-de-microsoft-robotics-studio/duda-sobre-el-color-de-un-nuevo-objeto-inse/view.html
- Simulación del Mudo con Microsoft Robotics Studio, Sara Morgan, 2009. http://msdn.microsoft.com/es-es/magazine/cc546547.aspx
- Concurrency and Coordination Runtime, Jeffrey Richter, 2009. http://msdn.microsoft.com/en-us/magazine/cc163556.aspx
- Tutorials and samples, 2009. http://msdn.microsoft.com/en-us/robotics/aa731536.aspx

10. ANEXOS

9. ANEXOS

9.1. Planificación

Este proyecto se ha realizado en siete meses, comprendidos del uno de Octubre de 2008 al 17 de abril de 2009. El tiempo estimado empleado por día es de seis horas y media. A continuación se muestra un resumen de tareas, horas y tiempo de ejecución.

Ésta planificación no es real, es una aproximación del reparto de tareas de las horas totales del proyecto.

DESCRIPCIÓN DE TAREA	DURACIÓN TAREA (horas)	FECHA DE EJECUCIÓN (desde - hasta)	OBSERVACIONES
1. Determinación del alcance del proyecto	8	06-10-08 al 07-10-08	
2. Construcción de línea de vista	8	08-10-08 al 09-10-08	
3. Aprendizaje MRDS	45	13-10-2008 al 21-10-2008	
4. Aprendizaje Programación asíncrona	110	22-10-2008 al 21-11-2008	Realización de tarea conjunta con 5
5. Desarrollo Documentación I	80	27-10-2008 al 04-12-2008	
6. Aprendizaje VPL	115	05-12-2008 al 12-12-2008	Se realizó un trabajo dirigido previo al proyecto de donde se adquirieron los conocimientos

			necesarios.
7. Aprendizaje Visual C#	50	15-12-2008 al 23-12-2008	
8. Diseño aplicación I (Entorno virtual)	100	05-01-2009 al 26-01-2009	Realización de tarea conjunta con 9.
9.Aprendizaje SolidWorks	75	09-01-2009 al 19-01-2009	
10.Pruebas I (Entorno virtual)	30	27-01-2009 al 02-02-2009	
11.Desarrollo de documentación II (Entorno virtual)	10	03-02-2009 al 06-02-2009	Se combina con el ajuste del entorno tras la fase de pruebas I.
12.Diseño de aplicación II (diseño y desarrollo de clases)	70	09-02-2009 al 23-02-2009	De las 70 h, 40 corresponden al diseño de clases y 30 al desarrollo.
13.Diseño de aplicación III (Movimiento del robot)	90	24-02-2009 al 24-03-2009	Realización de tarea conjunta con 14.
14. Pruebas II (movimiento)	70	12-03-2009 al 24-03-2009	
15. Desarrollo de documentación III (movimiento, pruebas, conclusiones…etc.)	30	25-03-2009 al 31-03-2009	
16.Diseño documentación	30	01-03-2009 al 07-04-2009	Se refiere a estructura de la memoria, maquetación…etc.
17.Diseño presentación	18	13-04-2009 al 16-04-2009	

9.2. Presupuesto

En la siguiente tabla se muestra el presupuesto detallado que supondría la realización del proyecto. Se detalla el precio y las horas específicas para cada tarea y los totales correspondientes suponiendo unos costes/hora hipotéticos, que tendrían que adaptarse dependiendo del mercado (no valdrían para una oferta comercial).

FASES	TAREA	HORAS	EUROS/ HORA	EUROS
PLANIFICACIÓN	Determinación del alcance del proyecto	8	15,00	120
	Construcción línea de vista de las actividades	8	15,00	120
APRENDIZAJE	MRDS	45	18,00	810
	SolidWorks	75	18,00	1.350
	Visual C#	50	18,00	900
	VPL	115	15,00	1.725
	Programación asíncrona y concurrente	110	18,00	1.980
DISEÑO	Clases	40	25,00	1.000
	Presentación	18	25,00	450
	Documentación	30	25,00	750
DESARROLLO	Aplicación	220	30,00	6.600
	Documentación	120	20,00	2.400
	Pruebas	100	25,00	2.500
TOTAL		**939**		**20.705**

9.3. Contenido del CD

A continuación se detalla el contenido del CD:

- Memoria
- xml: carpeta que contiene los archivos correspondientes al entorno del hotel, hotel.xml y hotel.manifest.xml.
- Follower: Carpeta que contiene todo lo referente a la aplicación, código fuente, archivos .xml, mundo virtual...etc. A continuación se explica brevemente los ficheros principales:
    - Follower. sln: Es el archivo de solución de Visual Studio desde donde se lanza la aplicación desarrollada.
    - Data.txt: fichero de texto donde se detallan todas las coordenadas de los nodos del grafo del hotel, así como nombre y distancia entre ellos.
    - FollowerSim.manifest es el fichero de manifiesto (configuración) que especifica los servicios que se van a ejecutar.
    - HotelAbril.Manifest: contiene la descripción del mundo simulado, todos sus componentes, posiciones, texturas...etc.
    - Media: en esta carpeta se encuentran todos los modelos 3D en formato .obj y .bos correspondientes a los objetos del mundo simulado.
    - Clases implementadas:
        - AdjacencyList.cs: mantiene una lista de los nodos vecinos para un nodo en particular.
        - Coordenadas.cs: Clase para crear objetos de tipo coordenadas, descritas en x,y, z.
        - CalculaRuta.cs: Clase que se utiliza para calcular la ruta óptima entre un nodo y otro utilizando el algoritmo de Dijkstra.

- EdgeToNeighbor.cs: Representa la arista que nace de un nodo hacia su nodo vecino.
- Graph.cs: Modela un grafo compuesto por una colecciÛn de nodos y aristas.
- Node.cs: Entidad utilizada para modelar cada uno de los puntos del grafo del hotel, de donde se pueden obtener las coordenadas, el nombre y los nodos vecinos.

9.4. Glosario de términos técnicos

- MRDS: Microsoft Robotics Developer Studio
- WFC: Windows Communication Framework
- VPL: Visual Programming Language
- VSE: entorno de simulación virtual de Microsoft
- CLR: Common Language Runtime
- DSS: Descentralized Software Services
- REST: Representational State Transfer
- DLL: Dynamic Linking Library
- HTTP: Hyper Text Transfer Protocol
- DSSP: Descentralized Software Services Protocol
- SWIFT: SolidWorks Intelligent Feature Technology
- LRF: Laser Range Finder
- FOV: Field of View
- LMS: Laser Measurement Sensor

- SRGS: Speech Recognition Grammar Specification